ioProgrammo

LA RIVOLUZIONE DI JAVA 5
NEL CD L'AMBIENTE DI SVILUPPO COMPLETO!

VERSIONE PLUS ✔ RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD ☐ RIVISTA+CD €6,90

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • NOVEMBRE 2004 • ANNO VIII, N.10 (85)

SMS GRATIS DAL TUO CELLULARE

Costruiamo un'applicazione J2ME che consente di inviare SMS via GPRS

- Come realizzare l'applicazione client e il componente server
- Tutto il codice commentato riga per riga
- Nel CD l'eseguibile Java da installare subito sul telefonino

Penetrare nei PC con una JPEG inviata da remoto!

Esclusivo: l'autore dell'exploit dell'anno ci svela i segreti per sfruttare la clamorosa falla di Windows

SOFTWARE ESTENSIBILE

Sviluppare applicazioni pronte per i plug-in

JAVA DISEGNA LE INTERFACCE

Trasformare in codice Java le form di Visual Basic

IN ESCLUSIVA

DA NON PERDERE

Visual Basic Power Pack
Sette nuovi controlli completamente gratuiti

SISTEMA
- XAML: ora le interfacce le disegno in Photoshop!
- Interpretare WordML e pubblicare via RSS
- Java Swing in pratica

ELETTRONICA
- Collega un radiocomando al computer

INTERNET
- Web Services in Java: un esempio completo
- WSE 2.0: il futuro dei servizi Web in .NET
- Pubblicare pagine per i-mode

CORSI
- ECLIPSE: le interfacce grafiche con SWT
- JAVA: software più robusto con le *eccezioni*
- C#: attributi e puntatori
- VISUAL BASIC: le nostre interfacce in stile XP

ISSN 1128-594X
ioProgrammo Anno VIII - N° 10 (85) • € 9,90
EDIZIONI MASTER www.edmaster.it

iPOD: UN'APPLICAZIONE CHE RECUPERA I DATI DAL PLAYER

CONTENTS ▼

Questo mese su ioProgrammo

Anno VIII - n. 10 (85) Novembre 2004

▼ Rovescio d'autore

È dell'ultima ora la notizia che Kodak ha chiesto un risarcimento di 1000 milioni di dollari (!) a Sun per la violazione di alcuni suoi brevetti nel codice di Java. Mi chiedo: quanto dovrà andare avanti questa guerra alla ricerca del codicillo? Siamo davvero sicuri che leggi restrittive sul diritto d'autore siano la migliore garanzia per un effettivo progresso tecnologico ed economico? Non dimentichiamo che è questo il fine ultimo della tutela del diritto d'autore: favorire la ricerca e lo sviluppo. Una tutela eccessiva comincia invece a diventare un freno.
ioProgrammo non ha mai fatto guerre di religione, a favore o contro l'Open Source, eppure il moltiplicarsi di azioni legali e di processi invita ad una riflessione più profonda. Il caso di Sun, che passa da carnefice (vedi il caso Microsoft) a vittima di questo sistema, è emblematico della oggettiva difficoltà cui si va incontro nell'operare nella programmazione senza incorrere in qualche infrazione legale.
Restando in Italia, le leggi sul diritto d'autore sono solo una piccola parte rispetto a quelle cui deve prestare attenzione uno sviluppatore che lavora in proprio o un'azienda informatica: i dubbi su cosa sia possibile distribuire e attraverso quale licenza sono moltissimi e sono in molti a chiedere un aiuto. ioProgrammo non può cambiare le leggi. Proveremo a spiegarle: non mancate i prossimi appuntamenti.

Raffaele del Monaco

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **giancarlo** *Password:* **cromalin**

ioProgrammo

Anno VIII - N.ro 10 (85) - Novembre 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Massimo Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Domenico Pingitore, Thomas Zaffino
Collaboratori R. Allegra, M. Autiero, A. Baldini, L. Barbieri, L. Buono, E. Florio, F. Grimaldi, M. Locuratolo, L. Mattei, P. Martemucci, S. Meschini, F. Mestrone, G. Naccarato, C. Pelliccia, P. Perrotta, M. Poponi, L. Spuntoni, F. Vaccaro, D. Visicchio.
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona
Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.p.a.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic (11 numeri) €52,90 sconto 30% sul prezzo di copertina di €75.90 ioProgrammo con Libro (11 numeri + libro) €86,90 sconto 30% sul prezzo di copertina di €123.90
ioProgrammo Basic + Internet Magazine Go!OnLine(22 numeri + 22 CD-Rom) € 64,90 anziché € 129,80
ioProgrammo con Libro + Internet Magazine Go!OnLine(22 numeri + 22 CD-Rom + 11 Libri) € 89,90 anziché € 177,80)
Offerte valide fino al 30/11/04
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore- zona ASI - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Ottobre 2004

Edizioni Master edita: Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, TV e Satellite, Win Extra, Home entertainment, Digital Japan, Digital Music, Horror Mania, ioProgrammo Extra, Le Collection.

News

JPEG: UNA BRUTTA FIGURA PER MICROSOFT

L'ultimo ciclone che ha colpito Microsoft riguarda le immagini JPEG e ha creato un vastissimo allarme in tutto il mondo.
Un exploit che sfrutta una falla di GDI+ e che ha trasformato in realtà l'incubo degli utenti Windows: la possibilità che anche le immagini si trasformino in vettori di virus.
Protagonista dell'exploit è stato uno dei migliori collaboratori di ioProgrammo, Elia Florio, intervistato per l'occasione da numerosi magazine americani. Tutti i dettagli a pagina 64!

DVD DA 1 TERABYTE

Questa è la promessa di un gruppo di ricercatori dell'Imperial College di Londra: stanno mettendo a punto una tecnologia che entro dieci anni porterà al grande pubblico supporti da 1000 Gigabyte. Con una capacità equivalente a 100 degli attuali DVD, i nuovi supporti avrebbero la stessa dimensione e sarebbero compatibili con gli attuali lettori. Il nuovo medium si chiamerà MODS (Multiplexed Optical Data Storag) e trarrà la sua forza dalla capacità di riconoscere 300 variazioni per pit (l'unità di scrittura del DVD) a fronte dell'attuale capacità di distinguere fra due soli stati (0 e 1). Già questa singola modifica, comprendendo la correzione di errore, porterà da sola un fattore di 10 nella capacità di immagazzinamento delle informazioni.

www.ic.ac.uk

OQO: IL PC IN PALMO DI MANO

La compagnia americana OQO, con quasi due anni di ritardo, ha annunciato di essere pronta a lanciare sul mercato un PC "full-optional" delle dimensioni di un Pocket PC: OQO model 01.

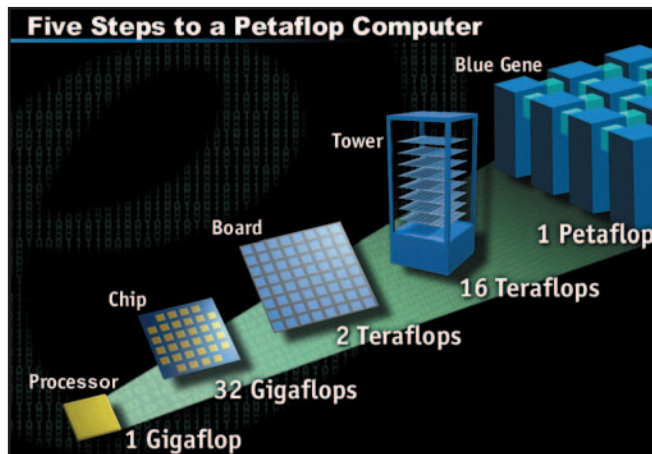
Queste le caratteristiche salienti: processore Transmeta da 1GHz, Hard Disk da 20GB, 256MB di RAM, schermo retroilluminato 800x480, pieno supporto per la connettività wireless (Wi-Fi, Bluetooth), porta USB e FireWire, microfono integrato. L'interattività è garantita dal comodo touchscreen. Il peso si aggirerà sui 400 grammi. Non c'è che dire: un bel gioiellino. Il prezzo dovrebbe essere compreso fra i 1500$ e i 2000$.

www.oqo.com

NUOVO RECORD MADE IN USA

Erano ormai due anni che l'America provava a riappropriarsi della palma di computer più veloce del mondo. Nel 2002 la giapponese NEC aveva infatti strappato il primato con un cervellone capace di 36,01 teraflop, il celebre Earth Simulator. È IBM a riuscire nell'impresa con un nuovo supercomputer che, sebbene ancora in versione beta, ha già raggiunto i 35,86 teraflop. Blue Gene, questo il nome del nuovo supercomputer, si avvale di un'architettura rivoluzionaria rispetto ai suoi predecessori: una tipologia che IBM definisce

Nodi	65,536
CPU per nodo	2
CPU totali	131,072
Velocità delle CPU	667 MHz
Prestazioni teoriche	360,000 GFlop/s
Memoria per nodo	512 Mega Byte
Memoria totale	32 Tera Byte
Tipo di memoria	DDR DRAM
Memoria su disco	800 Tera iBytea

SmsSender.zip

DA MICROSOFT UN LINGUAGGIO DI PROGRAMMAZIONE VISUALE DEDICATO AI ROBOT

Microsoft, in occasione di un evento svoltosi in Belgio, ha presentato il nuovo framework chiamato *Visual Robot Development Kit* (VRDK).
Il VRDK contribuirà a ispirare i futuri ingegneri consentendo loro di programmare con facilità dei robot "giocattolo" che siano capaci di compiere semplici compiti e che possano essere controllati attraverso uno smartphone basato sul sistema operativo Windows Mobile", ha dichiarato Microsoft in un comunicato.

www.research.microsoft.com

"a celle", e che consente un risparmio di energia e di spazio di circa due ordini di grandezza. Basato su Linux, il progetto è nelle sue prime fa-

si di sviluppo e fa attualmente affidamento su 16.000 processori, mentre, nella fase finale, potrà contare su 131.000 con l'obiettivo di superare i 360 teraflop.

www.research.ibm.com/bluegene

UN NEGATIVO PER LE FOTOCAMERE DIGITALI

Adobe sta spingendo perché le case produttrici di macchine fotografiche digitali adottino un formato unificato, al posto degli attuali RAW. DNG, Digital NeGative, è stato progetttato sulla base di TIFF EP, il medesimo che si trova alla base dei RAW attualmente utilizzati e che tanti problemi di incompatibilità portano. Il formato RAW è infatti direttamente legato al sensore utilizzato dalla macchina fotografica dunque: tutte le informazioni raccolte dal sensore si ritrovano dunque nel RAW, preservata da qualsiasi manipolazione. Questa caratteristica ha fatto del RAW il miglior formato per macchine di fascia alta e professionali, a scapito però della compatibilità. DNG si propone di essere l'uovo di colombo: massima qualità all'interno di un formato standard.

www.adobe.com/dng

UN SUPERCOMPUTER PER SUN

Sun Microsystem e l'Università del Texas hanno rivelato di essere pronti ad avviare un supercomputer per l'analisi di grosse masse di dati. Maverick è il nome del nuovo cervellone che, basato su server Sun Fire E25K, sarà utilizzato nel centro di ricerche UT Austin's Texas Advanced Computing Center. I dettagli finanziari del progetto non sono stati svelati anche se l'ordine di grandezza si può supporre di qualche milione di dollari. Maverick riunisce il meglio della tecnologia sia nella visualizzazione dei risultati che nello scambio di dati via rete, indispensabile per poter accedere all'enorme quantità di dati da elaborare, ad esempio in campo meteorologico. Il sistema utilizza 64 processori UltraSparc 4 ed ha a disposizione 512 gigabyte di memoria condivisa. Ovviamente, il sistema operativo scelto è Solaris.

www.sun.com

PALM OS A CACCIA DI SMARTPHONE

PalmSource, produttore di Palm OS, ha annunciato la disponibilità di un nuovo sistema operativo progettato specificamente per smartphone: Palm OS Cobalt 6.1 Le incerte prospettive di mercato dei PDA hanno spinto PalmSource verso quello che promette di essere l'eldorado dei prossimi 5-10 anni: i telefonini di fascia alta. Il nuovo sistema operativo supporta nativamente Wi-Fi e Bluetooth e, offre la più vasta scelta per la gestione della connettività: Bluetooth, memory card SD, USB. Sono attesi per la fine dell'anno i primi dispositivi con installato il nuovo sistema operativo.

www.palmsource.com

FLEXWIKI: ANCORA OPEN SOURCE PER MICROSOFT

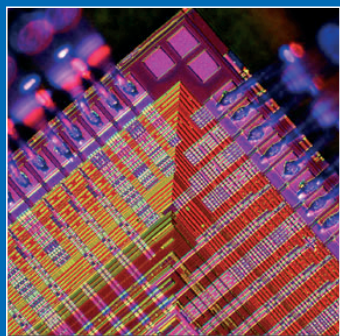
La tentazione verso il software Open Source è sempre più forte per Microsoft: questa volta tocca a FlexWiki, un sistema di collaborazione per la creazione di siti "cooperativi". Il modello è quello di WikiPedia, per intenderci, dove chiunque può pubblicare e modificare quanto pubblicato da altri. FlexWiki è scritto in C# e segue a breve distanza altri due progetti rilasciati da Microsoft come Open Source: Windows Installer XML e Windows Templater Library. Che sia un modo per capire le dinamiche dello sviluppo Open Source? Quel che è certo è che la comunità Open Source si arricchisce di un altro "pezzo pregiato".

http://sourceforge.net/projects/flexwiki

IL NUOVO 3D DI MICROSOFT

WGF 1.0 prenderà il posto delle Direct3D 10. Nella prossima release delle DirectX scomparirà la tecnologia Direct3D a favore delle nuove API Windows Graphics Fondation
Le Windows Graphics Fondation si integreranno appieno nella nuova tecnologia Avalon, parte del prossi mo sistema operativo Microsoft: Longhorn.

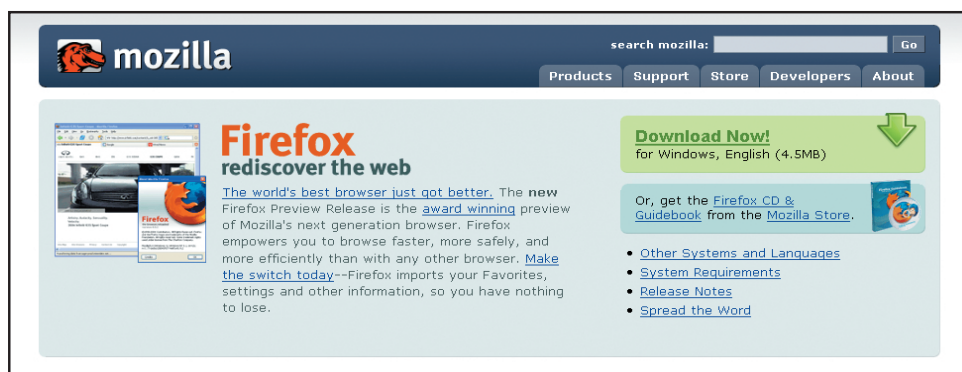
I PORTATILI SFIDANO I DESKTOP

Intel annuncia la disponibilità di un processore per notebook di fascia desktop-replacement con una frequenza di 3,3 GHz, molto vicina ai più veloci modelli Pentium 4.
Il nome del nuovo processore è Mobile P4 548 ed è caratterizzato da un consumo massimo di 88 watt, ad una tensione compresa fra 1,25 e 1,4 volt.

Il prezzo, per mille unità, è di circa 260 dollari. Il P4 548 supporta la tecnologia Hyper-Threading ed è costruita attraverso un processo a 90nm.
Offre 1MB di memoria per la cache di secondo livello ed è compatibile con i chipset Intel 852 GME e 852PM.

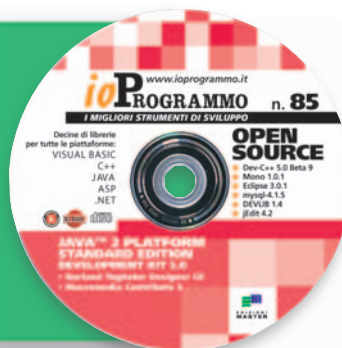
www.intel.it

FIREFOX 1.0 PREVIEW: PARTENZA A RAZZO!

Superando le aspettative della Mozilla Foundation che prevedeva di superare il milione di copie scaricate in dieci giorni, la versione preview di Firefox 1.0 è stata scaricata in cinque giorni da 1,3 milioni di persone. Dopo che Microsoft ha in pratica abbandonato lo sviluppo di Internet Explorer come prodotto stand-alone, Firefox si trova a "fronteggiare" l'entusiasmo di milioni di utenti che sembrano aver finalmente trovato un'alternativa valida al browser di casa Microsoft. IE, anche a causa della sua estrema diffusione, ha offerto il fianco a numerose critiche riguardo alla sicurezza, critiche che cominciano a toccare anche Firefox. "Da grandi poteri derivano grandi responsabilità": ora Firefox dovrà dimostrare di valere la posizione di sfidante ufficiale di IE. L'apprezzamento di cui gode Firefox poggia essenzialmente su quattro pilastri: stabilità, velocità, ricchezza di plug-in e "immunità" a virus e spyware che hanno afflitto negli ultimi anni IE. Proprio questo fronte dovrà essere curato maggiormente: pare infatti che cominciano a diffondersi i primi co dici malevoli contro Firefox. Si riaccende la guerra dei browser?

www.mozilla.org

UN'ALLEANZA PER GLI SMARTPHONE

Tre giganti del calibro di Nokia, Intel e Symbian hanno siglato un accordo di cooperazione per lo sviluppo di dispositivi su piattaforma 3G. Lo scopo è quello di creare una piattaforma di riferimento che unisca: i chip CScale di Intel, il sistema operativo di Symbian e l'interfaccia Series 60 di Nokia. Dato il peso dei tre, l'accordo potrebbe fare da polo gravitazionale per tutti gli altri costruttori di telefonini, raggiungendo uno standard de facto che, oltre a semplificare la vita agli utenti, solleverebbe molti produttori dagli oneri della ricerca.
È chiaro che Intel vorrebbe ripetere lo stesso modello di Business che l'ha vista trionfare nel mercato dei PC grazie alla partnership con Microsoft: l'obiettivo sembra essere traslare il modello Wintel nel mercato mobile.
Dopo aver già raggiunto lo status di prima scelta sia per gli smartphone basati su Windows Mobile, sia per quelli che montano Palm OS, Intel si conferma dunque il riferimento per il mobile.

SOFTWARE SUL CD

Borland Together Designer CE

La modellazione professionale per tutti gli sviluppatori

Un ambiente di modellazione gratuito di livello professionale per la creazione di diagrammi UML (*Unified Modeling Language*) standard. Questa release offre ad architetti, analisti aziendali e sviluppatori l'opportunità di godere dei vantaggi legati alla modellazione a costo zero. Tra le principali caratteristiche di *Together Designer Community Editing* segnaliamo:

- Creazione di diagrammi UML 2.0 e UML 1.4.
- Creazione di diagrammi in formato immagine *(*svg, *bmp* e **gif)*.
- XMI per l'importazione UML.
- Importazione di modelli **.mdl* di Rational Rose.
- Creazione di diagrammi ER logici.

L'INTERFACCIA

L'esperenzia di Borland la si trova riversata subito nel look&feel dell'applicazione che, attraverso un sapiente uso del colore e delle icone, riesce comunicare all'utente una

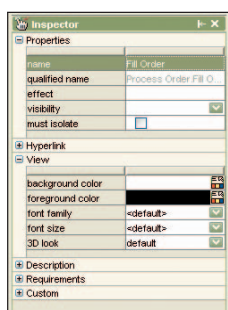



Fig. 2: **Con l'Inspector possiamo modificare qualsiasi elemento**

grande immediatezza, anche a dispetto della inevitabile complessità dell'ambiente. Tutti i componenti del progetto si posizionano con un semplice drag&drop, mentre le proprietà di ogni oggetto possono essere ispezionate e modificate con l'apposito pannello *Inspector*.

UML 2.0

Together consente di lavorare sia secondo lo standard UML 1.4 sia con il 2.0, inoltre risulta molto interessante la possibilità di convertire diagrammi UML 1.4 nella versione 2.0 attraverso un apposito wizard. La nuova versione di UML si arricchisce di nuovi diagrammi dedicati alla modellazione architetturale, aggiungendo anche alcune migliorie nei diagrammi già esistenti. Mentre gli *Use Case Diagrammi* restano pressoché identici, gli *Activity Diagram* sono stati profondamente rivisti: non più specializzati in una rappresentazione di automi a stati finiti, gli *Activiti Diagram* si pongono ora a metà strada fra i tradizionali *Data Flaw Diagram* (DTD) ed i flow chart. A voi scoprire le altre novità di UML 2 con *Togheter!* Oltre ai diagrammi UML nelle due versione, *Togehter* offre anche la possibilità di definire diagrammi Entità-Relazione in modo del tutto coerente con i progetti UML che sviluppiamo: ottima idea, considerando che è ormai difficile immaginare un progetto software che non includa un database.

Fig. 1: **Gli Activity Diagram sono fra le novità di UML 2.0**

Fig. 3: **Dal pannello Project si controlla l'intero progetto**

INSTALLAZIONE

L'installazione del software è semplicissima, in pratica basta un doppio clic sul file di setup. Il prodotto è gratuito e, al primo avvio, è richiesta una chiave di attivazione per ottenere la quale è necessario collegarsi al link *www.borland.com/products /downloads/download_together.html*, registrandosi al sito della Borland se non si è già iscritti. Ovviamente non è necessario scaricare il file di installazione dal sito, essendo già presente nel CD allegato alla rivista.

☑ Together Designer Community Edition
Produttore: Borland
Sul web: www.borland.it
Prezzo: Gratuito
Nel CD: *\TogetherCE*

Java 2 SE SDK 5.0

Alla scoperta delle novità di Tiger

Dopo anni di spasmodica attesa, finalmente possiamo mettere le mani sulla nuova versione del linguaggio più amato dai programmatori: nome in codice Tiger. Le nuove caratteristiche del linguaggio hanno una cosa in comune: prendono alcuni costrutti idiomatici usati generalmente dai programmatori e ne forniscono un supporto linguistico. In altre parole, è stata spostata dal programmatore al compilatore il compito dello scrivere il cosiddetto "boilerplate code", quelle porzioni di codice che si ripetono sempre immutate da un programma all'altro e che hanno l'unico scopo di introdurre errori… Grazie al fatto che il compilatore, a differenza del programmatore, non commette errori, il codice prodotto risulta essere meno soggetto a bug.
I principali miglioramenti:

Fig. 1: ***La struttura della nuova piattaforma***

- **Generics**: fornisce dei meccanismi di controllo a tempo di compilazione per le collection ed elimina la noia di effettuare il casting
- **Miglioramenti nei cicli**: avremo degli iteratori più semplici e più immuni da errori
- **Autoboxing/unboxing**: elimina la necessità della conversione manuale tra tipi primitivi (ad esempio int) e wrapped (Integer)
- **Typesafe enums**: tutti i noti benefici della struttura Typesafe Enum, senza la passata prolissità e la conseguente tendenza a causare errori
- **Import statico**: non sarà più necessario qualificare i membri statici con il nome della classe cui appartengono
- **Metadata**: il supporto per i metadati consentirà ai vari tool disponibili di generare codice automaticamente, a partire dalle annotazioni indicate nel codice.

L'innovazione risulta particolarmente interessante, perché porta ad uno stile di programmazione "dichiarativo", in cui il programmatore "dice" cosa deve essere fatto ed i tool di turno generano il codice che soddisfa la richiesta.

Produttore: Sun Microsysems
Sul web: www.sun.com
Prezzo: Gratuito
Nel CD: ***java 5 sdk***

Macromedia Contribute 3

L'aggiornamento dei siti "for dummies"!

Con questa nuova versione, il software di Macromedia permette anche agli utenti meno esperti di aggiornare le pagine di siti web o quelle della Intranet con la stessa semplicità con cui potrebbero generare un qualsiasi documento Word. Macromedia Contribute 3 dispone ora di un miglior sistema di controllo dell'amministrazione, di una serie di miglioramenti nell'editing e dell'integrazione con il sistema publishing di Dreamweaver MX 2004. Gli amministratori non devono fare altro che stabilire chi ha i permessi di pubblicazione e chi può effettuare solo editing. In questo modo il lavoro può essere ottimizzato e distribuito in maniera agevole e sicura.

Fig. 1: ***L'interfaccia di Contribute si presenta poco più complessa di un browser***

LE NOVITÀ

Le nuove funzionalità per la gestione dei contenuti includono un sostanziale miglioramento della performance, il supporto per l'editing di una tipologia di siti più ampia, la gestione e modifica delle immagini, l'integrazione con Microsoft Office e la pubblicazione automatica dei documenti in formato PDF, Flash e Flashpaper 2. Per i Web Designers e gli sviluppatori, Contribute 3 rende disponibile il supporto a WebDAV e incorpora il motore per il rendering dei CSS (*Cascade Style Sheet*) di Dreamweaver MX 2004. Un'inedita funzionalità consente la revisione e l'editing del codice sorgente della pagina Web realizzata attraverso un qualsiasi editor HTML, prima della pubblicazione. Per l'installazione è necessaria una chiave di attivazione che si può ottenere, via mail, collegandosi alla pagina HYPERLINK "*www.macromedia.com/downloads*" *www.macromedia.com/downloads*, cliccando sul link try, a fianco della voce Contribute 3.

☑ Contribute 3

Produttore: Macromedia
Sul web: www.macromedia.it
Prezzo: € 149
Nel CD: ***\Contribute 3***

Sothink SWF Decompiler MX 2005

di Maurizio Battista

Un software per decompilare con facilità i filmati Flash

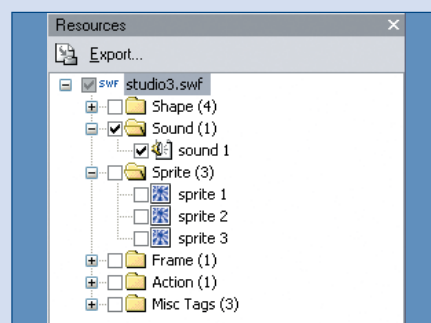
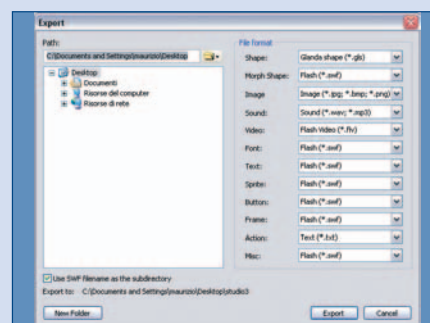
Lo scorso 30 luglio, la SourceTec Software ha rilasciato la versione MX 2005 del più celebre tra i software dedicati alla riconversione dei file SWF: Sothink SWF Decompiler. Per chi non ne avesse mai sentito parlare, è un applicativo in grado di decompilare un filmato Flash precedentemente pubblicato per il web, risalendo al corrispondente file FLA. Indubbiamente questa procedura viaggia sulla sottile linea di confine situata tra il lecito e la pirateria. Un utilizzo onesto del programma consente di recuperare propri lavori nel caso si siano smarriti i file nativi - una procedura completamente legale, in grado di rimediare a situazioni altrimenti irrisolvibili. Sarebbe illecito, invece, ricostruire un file sorgente altrui senza alcuna autorizzazione. Tralasciando questi aspetti, che rientrano nella complicata questione del diritto d'autore, dal punto di vista tecnico Sothink SWF Decompiler permette di importare un file SWF e smontarne i vari elementi.

Fig. 1: *L'ambiente di lavoro di Sothink SWF Decompiler MX 2005*

L'AMBIENTE DI LAVORO

L'ambiente di lavoro si presenta come una schermata ripartita in varie sezioni. Nella parte superiore si trova la barra che contiene i menu *File, Edit, View, Code, Option* e *Help,* mentre sotto è collocata la Toolbar, con le icone corrispondenti alle funzionalità più importanti. La sezione a sinistra corrisponde all'*Explorer panel* per navigare tra le cartelle del proprio sistema e individuare i filmati SWF presenti. Al centro si trova la *Preview Window,* che in base alle impostazioni predefinite visualizza un'anteprima del filmato SWF in azione come se fosse letto dallo Stand Alone Flash Player. Nella parte inferiore dell'area di lavoro, la sezione *Information Window* fornisce una serie di informazioni aggiuntive sull'elemento selezionato. A destra è visibile il *Resource Panel,* nel quale appaiono, suddivisi in varie cartelle, tutti gli elementi che compongono il filmato: immagini bitmap, suoni, video, simboli statici, pulsanti, clip filmati e script. Effettuando la selezione di un elemento nel *Resource Panel,* automaticamente la *Preview Window* si aggiorna mostrandone il contenuto all'utente: per esempio, se nella cartella *Action* selezioniamo un codice, questo sarà visualizzato in anteprima nella sezione centrale della finestra di lavoro. Una volta "frammentati" gli elementi del file SWF, Sothink SWF Decompiler consente di recuperarli separatamente, per poi collocarli, uno alla volta, in un nuovo filmato Flash. Una procedura alternativa si traduce nell'uso di una funzione automatica che ricostruisce per intero il file sorgente con estensione .FLA.

ESTRARRE UN FILE MP3

1 **Apriamo il programma e, adoperando l'*Explorer panel*, selezioniamo il file *studio3.swf* disponibile nella cartella degli esempi. In alternativa, preleviamo il file utilizzando l'apposito pulsante nella barra degli strumenti di Explorer.**

2 **Nella sezione a destra (*Resource Panel*) appare *studio3.swf*. Facciamo clic sulla icona a forma di (+) in modo da accedere alle sottocartelle. Espandiamo la cartella *Sound* e selezioniamo *Sound1*.**

3 **Premiamo il pulsante *Export* che si trova in *Resource Panel*: l'elenco dei file che il programma è in grado di esportare appare nella finestra *Export*, in cui possiamo scegliere dove collocare il file estratto. Per gli esperimenti, potete utilizzare i file presenti nel CD: *\Codice\sorgenti_fla.zip***

In tal caso potrebbero esserci degli elementi che il programma non riesce a ricostruire correttamente: per esempio i nomi dei campi di testo dinamici e di input, e alcuni tipi di istruzioni. Per ovviare a questo inconveniente, si può solo procedere a una ricostruzione ragionata degli elementi mancanti, basandosi sulla logica del progetto. Infatti, anche se non riesce a ricostruire determinate istruzioni presenti nel codice ActionScript (in particolare le nuove classi), il programma le visualizza comunque nella *Preview Window*. Se Internet Explorer è il browser predefinito sul nostro computer, dopo che Sothink SWF Decompiler è stato installato, nella barra degli strumenti appare il pulsante Sothink SWF Catcher. Facendo clic sulla sua icona si attiva la finestra *Save*, che consente di "catturare" i filmati SWF visualizzati durante la navigazione, salvandoli in un cartella di nostro gradimento. Grazie al pulsante *SWF Decompiler*, inoltre, dopo il salvataggio possiamo importare il filmato direttamente nel programma. Nel caso in cui la pagina che contiene il filmato SWF sia inserita in una finestra creata con JavaScript, è necessario aprire una nuova finestra vuota del browser e trascinare al suo interno l'icona di Internet Explorer: a quel punto sarà possibile fare ricorso al pulsante Sothink SWF Catcher. Se invece Internet Explorer non è il browser predefinito, per prelevare i file SWF incontrati durante la consultazione di pagine sul Web bisogna ricorrere a tecniche alternative (tra cui il classico ripescaggio dalla cache del computer). La versione trial di Sothink SWF Decompiler, che consente un periodo di utilizzo di quindici giorni, non consente di ricostruire in modo automatico i file FLA completi e limita la consultazione a un numero massimo di due soli script.

```
23      for (i = 0; i < punti.length; i++)
24      {
25          attachMovie("clip", "clip" + i, i);
26          this["clip" + i].trascina();
27      } // end of for
28  } // End of the function
29  function conversione(istanza_clip)
30  {
31      var _l13 = istanza_clip.x;
32      var _l12 = istanza_clip.y;
33      var _l11 = istanza_clip.z;
34      var _l4 = Math.sqrt(_l13 * _l13 + _l12 * _
35      if (_l13 == 0)
36      {
37          var _l2 = 1.570795;
38      }
39      else if (_l13 != 0)
40      {
41          _l2 = Math.atan(_l12 / _l13);
42      } // end if
43      if (_l12 < 0)
44      {
45          _l2 = _l2 + 3.141593;
```

Fig. 2: ***Il codice mostrato nella Preview Window***

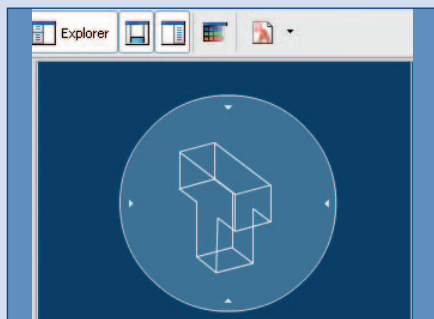
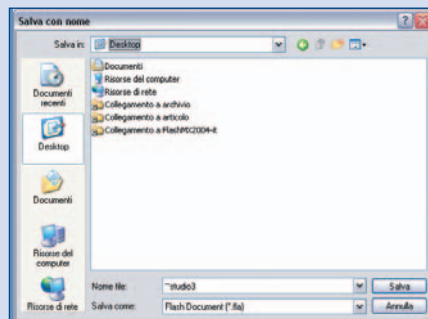
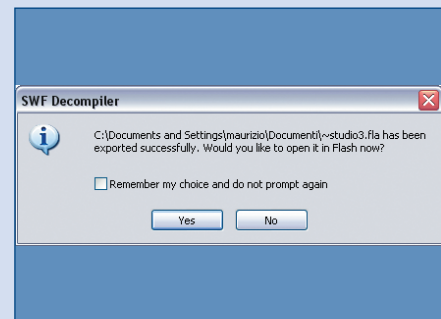
Fig. 3: ***il pulsante Sothink SWF Catcher***

LE CONTROMISURE

Per concludere, vediamo quali contromisure abbiamo a disposizione se vogliamo impedire che qualche "malintenzionato" adoperi il decompilatore per saccheggiare i nostri SWF. Le strade percorribili sono poche. Sicuramente non è possibile proteggere in alcun modo i contenuti grafici e sonori, anche se il file è esportato con l'opzione *Proteggi da importazione*. Gli unici due espedienti che si possono tentare sono i seguenti: offuscare il codice in modo da renderlo meno leggibile oppure ricorrere a un uso intensivo di classi associate ai clip filmati. Come tutti gli utenti di Flash sanno, a partire dalla versione MX 2004 è stata introdotta la possibilità di creare file con estensione .AS esterni che funzionano in modo simile alle classi java: se si prova a decompilare uno di questi file, pur visualizzando il codice della classe non si riesce a riprodurre un file FLA completo e funzionante. Questa soluzione potrebbe non risolvere del tutto il problema. Infatti, se chi usa il decompilatore è un programmatore esperto, leggendo le classi mostrate nella sezione centrale potrebbe ricrearle per il nuovo FLA esportato. Tuttavia, poiché non è un'operazione semplice o immediata, diventa un modo per rendere la vita difficile a quanti decidessero di usare Sothink SWF Decompiler con un nostro lavoro.

☑ Sothink SWF Decompiler MX 2005

Produttore: SourceTec Software
Sul web: www.sothink.com
Prezzo: $ 79.99
Nel CD: *swfdec.zip*

RICOSTRUIRE UN FILE FLA

1 Apriamo il file *studio3.swf*. Se il programma riesce a eseguire in anteprima tutte le funzioni dell'SWF, abbiamo già un buon indizio sull'esito positivo dell'esportazione automatica. In alcuni casi, Sothink SWF Decompiler non riesce a esportare un FLA funzionante.

2 Selezioniamo il pulsante *Export FLA*, (la seconda icona da sinistra nella Toolbar). Così facendo il programma apre la finestra *Salva* con nome, nella quale per impostazione predefinita crea un file FLA chiamato *~studio3.fla*.

3 Appare una finestra in cui ci viene chiesto se vogliamo aprire il file appena salvato con Flash. Facendo un raffronto, notiamo che Sothink SWF Decompiler rinomina i simboli della libreria e sostituisce i commenti presenti nel codice.

Xamlon 0.9

Prova in anteprima XAML

Lo sviluppatore Paul Colton ha messo a disposizione un software xamlon (*www.xamlon.com*) che consta di due strumenti: Visual Designer for Visual Studio .NET 2003 e XamlPad. Il primo mette a disposizione un plug-in per Visual Studio .NET 2003, che permette di salvare l'interfaccia grafica di un'applicazione in linguaggio XAML; il secondo implementa, invece, una sorta di Notepad per scrivere e testare script XAML; in aggiunta consente finanche di convertire file SVG e C# in linguaggio XAML nativo. L'installazione del pacchetto prevede che sul proprio sistema operativo sia correttamente installata la versione 1.1 del .NET Framework.

Code Counter Pro 1.2

Un contachilometri per chi programma!

Una comoda applicazione che consente a noi programmatori di quantificare il lavoro che abbiamo svolto o stiamo svolgendo. *Code Counter* si occupa di valutare il numero di righe, spazi bianchi, commenti e quant'altro sia interessante per effettuare statistiche sul codice. I linguaggi supportati sono numerosi: C/C++, Java, Delphi/ Pascal, VB, PHP, ASP ed altri ancora, ed è possiblile effettuare analisi anche su porzioni di testo che non siano riconducibili a codice sorgente.

ccountp.exe

Snippet Compiler 1.11

Compilatore per snippet in C#

Quante volte vi è capitato di dover generare una soluzione .NET solo per testare poche righe di codice? Snippet Compiler evita proprio questo genere di "fastidio": attraverso questo piccolo e leggerissimo IDE potremo compilare le nostre porzioni di codice (in C#, VB.NET e ASP.NET) senza dover lanciare il più potente e ingombrante Visual Studio. Benché si tratti di una realizzazione "artigianale" Snippet Compiler compie il suo lavoro alla perfezione.

SnippetCompiler.zip

ColorCache 2.0.1

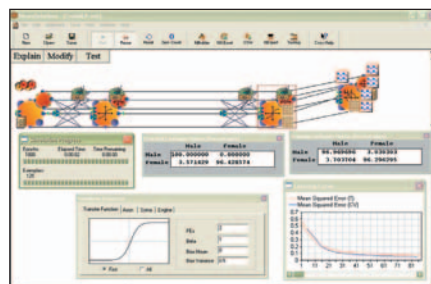
Un aiuto nella scelta dei colori

Un tool completo e semplice da utilizzare che ci aiuta nel comporre l'interfaccia delle nostre applicazioni. Possiamo individuare (con un "contagocce") il colore presente in una zona qualsiasi dello schermo e, a partire da un qualsiasi colore, possiamo farci indicare quali sono i complementari, i più simili, quelli che maggiormente contrastano e così via. Versione di prova valida trenta giorni.

cche2010.exe

NeuroSolutions 4.32

Progettare e simulare reti neurali

Un interessante esempio di come si possa arrivare, per via grafica, alla realizzazione di complesse reti neurali. L'interfaccia grafica è molto semplice e piacevole da utilizzare: attraverso un ampio utilizzo di elementi modulari, NeuroSoluions riesce a diluire la difficoltà insita in progetti di questo tipo. Numerosi sono i grafici con cui è

possibile visualizzare l'attività della rete e gli esempi disponibili aiutano anche il profano ad entrare nel mondo delle reti neurali con relativa semplicità. Ottimo il Wizard che consente di creare una libreria DLL a partire da una rete progettata con NeuroSolution. Versione dimostrativa valida sessanta giorni.

nsinstall.exe

WhosOn 3.1

Statistiche per i tuoi siti

WhosOn consente di monitorare l'attività sui siti che gestiamo, differenziando fra la presenza di spider e quella di utenti online. Analisi statistiche basate sulle serie storiche consentono un migliore lettura dei dati e, grazie alla chat integrata, è possibile dialogare a distanza con gli amministratori dei server remoti. Interessante la funzione di Hacker detection che ci avvisa in caso di accessi "anomali" al server. Versione dimostrativa valida trenta giorni.

WhosOn30.exe

Advanced PDF to HTML Converter 1.6

Converte documenti PDF in HTML

Molto efficace questa utility che si occupa di convertire documenti PDF in pagine HTML, preservandone la formattazione e consentendone la fruizione via Web. Non richiede la presenza di Acrobat Reader per funzionare e può essere istruito per effettuare più conversione in batch. Versione dimostrativa: su tutte le pagine prodotte è impresso un Watermark.

pdf2html_1_6_setup.exe

Fastream NETFile FTP/Web Server 6.9.5

Server FTP e Web in un pacchetto

Un server gratuito e decisamente spartano che consente l'amministrazione da remoto grazie ad una semplice interfaccia grafica. Autenticazione, impostazione differenziata della velocità di upload e di download, definizione delle quote di storage, gestione dinamica dei DNS: sono alcune delle caratteristiche di questo piccolo e valido strumento. Gratuito.

NetFileServer.exe

Deep Log Analyzer 1.4

Per raccogliere statistiche sul Web

La visualizzazione per via grafica delle statistiche è il punto di forza di Log Analyzer: il numero e la complessità delle diverse analisi possibile consentono di avere in ogni momento il quadro aggiornato dell'accesso ai nostri siti. I raggruppamenti che abbiamo a disposizione vanno a definire un ampio campionario di possibili report mentre la grafica evoluta consente una efficace interattività con l'utilizzatore. Versione valida venticinque giorni.

dlatrial.exe

Xlight FTP Server 1.6c

Un server FTP tutto compreso

Molto semplice da installare e da utilizzare, questo server FTP non rinuncia a funzionalità avanzate come il supporto per ODBC e per SSL. I numerosi settaggi comprendono la possibilità di definire gruppi di utenti cui riservare speciali aree, l'opportunità di definire "banned" alcuni IP, insieme ad una lunga serie di regole di sicurezza. Versione di valutazione valida trenta giorni.

setup.exe

XML:Wrench 1.2

Manipolazione di file XML

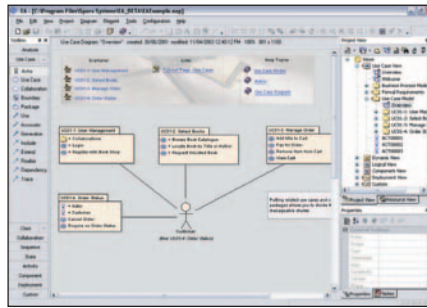
Un editor XML decisamente spartano che fa della semplicità il suo punto di forza. Con XML:Wrench è possibile manipolare anche file HTML e XHTML, il tutto senza inutili fronzoli e con un'interfaccia che piacerà agli utenti più smaliziati e meno

avvezzi alle funzionalità che rallentano la macchina prima ancora di velocizzare il lavoro dello sviluppatore! Sono comunque disponibili alcune indispensabili funzionalità come l'auto-completion.
La migliore caratteristica: è gratuito.
xmlwrench-v120.exe

Resource Standard Metrics 6.52
Analizza il tuo codice Java, C e C++
Resource Standard Metrics (RSM) offre un ricco ventaglio di analisi utili a definire la qualità del tuo codice. E possibile verificare che il codice che produciamo rispetti oltre cinquanta regole comunemente accettate come indice di qualità. La precisione e la velocità di questo prodotto hanno fatto sì che fosse scelto da oltre 1000 aziende in tutto il mondo. Versione di prova, analizza un massimo di dieci file.
rsm.zip

TextPipe Pro 7.0.7
Manipolazione di testi
Una eccellente soluzione per quella vasta schiera di sviluppatori che vanno dai programmatori ai web-designer e che ha necessità di portare a termine complesse operazioni di manipolazione su file testuali. TextPipe include moltissimi filtri utili a ottimizzare l'aspetto di codice sorgente scritto in qualsiasi linguaggio. Sofisticato nelle funzionalità ma semplice da utilizzare. È stato rinnovato il Wizard che guida alla ricerca testuale all'interno di tag HTML. Trenta giorni di prova.
textpipepro-cn.exe

WebCompiler 2.1.16
Da HTML a EXE: ora e facile!
Se vi trovate nella necessità di pubblicare un e-book, un manuale o la demo di un sito, questo è il prodotto che fa per voi: WebCompiler riesce a convertire, in breve tempo e con un intervento minimo da parte dello sviluppatore, le pagine HTML di un sito in una completa applicazione stand-alone. Con il supporto per JavaScript, Flash e cookies, offre una intelligente soluzione ad un ampio ventaglio di problematiche.
x2nwc.zip

Enterprise Architect 4.1
Per sviluppare e documentare software Object Oriented
Un completo ambiente visuale per sviluppare e manutenere software object oriented. Con il pieno supporto di UML 2.0, e di tutti i diagrammi contemplati dallo standard. Enterprise Architect consente di verificare l'integrità delle dipendenze fra i vari oggetti che compongono i nostri progetti e permette di modellare la gerarchia delle classi, sia staticamente sia dinamicamente. Con Enterprise Architect è possibile documentare con precisione e rapidamente qualsiasi progetto di sviluppo software, curando tutte le fasi del ciclo di vita del progetto: dalla ideazione alla confezione finale, passando per la fase di test e del controllo sui cambiamenti.
Versione di valutazione valida trenta giorni.
easetup.exe

Neuron PE Disassembler 1.0 b3
Disassemblare DLL, OCX ed EXE
Un tool che, attraverso una interfaccia visuale, consente di leggere il codice di librerie DLL e componenti OCX, oltre che dei file eseguibili Win 32. La lettura del codice è aiutata dalla struttura ad albero che visualmente viene resa sullo schermo. Versione dimostrativa valida trenta giorni.
neuronpedisassembler.zip

JDebugTool 3.7
Un debugger Java full optional!
Un debugger stand alone costruito sulla base della JPDA, Java Platform Debugger Architecture. L'interfaccia grafica ed un Help ottimamente strutturato consentono un rapido utilizzo del tool. Tra le funzioni segnaliamo: debug remoto, debug multithread, modifica delle variabili "al volo", visualizzazione delle classi attualmente caricate, valutazione dei *toString*, monitoraggio dell'occupazione di memoria, sincronizzazione con gli eventi di *load* e *unload* delle classi. In questa versione si apprezza particolarmente la velocità di esecuzione. Demo valida quindici giorni.
debugtool_sdk14.jar

Neuron Developer Studio 1.0b3
Disassembla file .CLASS e .EXE
Un tool che aiuta lo sviluppatore nella gestione delle risorse a sua disposizione. Eseguibili Java e Win32 possono essere esplorati attraverso una interfaccia grafica che ne evidenzia gli oggetti e le classi che li compongono. Interessante anche il supporto verso i database che possono essere interrogati via ODBC. Versione dimostrativa valida trenta giorni.
neuronshell.zip

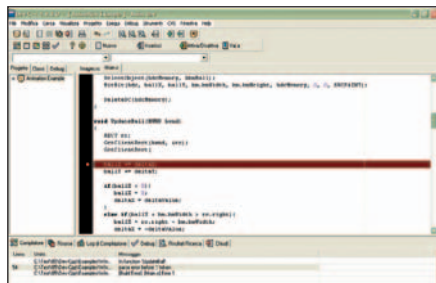
Batch File Compiler 2.0
Trasformare un file batch in EXE
La funzionalità principale di questa applicazione è tradurre in .EXE qualsiasi file batch (.bat). In aggiunta, abbiamo la possibilità di sfruttare alcune commodity come l'utilizzo dei colori e l'esecuzione di semplici operazioni matematiche. L'interfaccia grafica è quanto di più semplice si possa immaginare. All'atto della compilazione, è possibile scegliere come target il DOS 6.0 o Windows. Versione di prova, limitazioni su comandi utilizzabili.
bfcped.exe

CASE Studio 2 2.17
Diagrammi Entità-Relazione
Uno strumento che consente di progettare qualsiasi database attraverso i Diagrammi Entità-Relazione. L'interfaccia grafica consente di avere il pieno controllo del progetto ed è possibile generare il relativo DB per tutti i più diffusi DBMS: Oracle, DB2, MSSQL, Access, Sybase, InterBase, Firebird, MaxDB, MySQL, e PostgreSQL. I dettagliati report in HTML si rivelano di grande utilità all'atto della documentazione di un progetto. Versione dimostrativa limitata nelle funzionalità.
casestudio.zip

Code Charge Studio 2.3
Un contachilometri per il codice!
Una delle migliori soluzioni per la creazione di applicazioni Web database-driven per via visuale. Davvero ampio il supporto per tutti i più diffusi database e qualsiasi server Web. CodeCharge Studio rappresenta un'ottima soluzione per sviluppatori con qualsiasi esperienza (Access, progettisti Web, ecc.) per fare il grande salto e lanciarsi nella realizzazioni di complete soluzioni e-business.
CCStudio2_3.exe

Dev-C++ 5.0 Beta 9

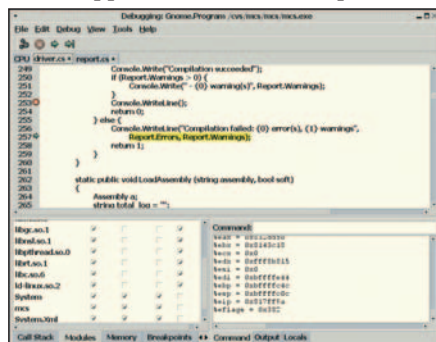
Un ottimo JDE

Ambiente integrato per lo sviluppo di applicazioni C/C++ distribuito con licenza Open Source GNU GPL.

Il programma utilizza Mingwn, port del potente compilatore GCC (GNU Compiler Collection), ma può essere utilizzato con Cygwin o qualsiasi altro compilatore basato su GCC. Dev-C++ consente di creare eseguibili Win32 come applicazioni GUI, console, librerie statiche e DLL. Gratuito, mediante l'installazione è possibile impostare come lingua l'italiano.

devcpp4990setup.exe

Mono 1.0.1

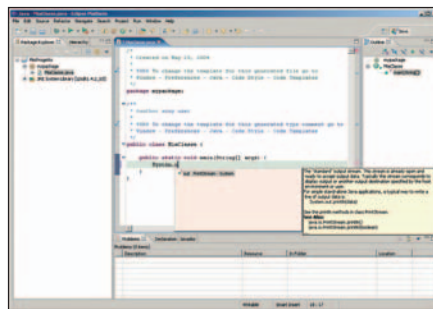
Framework .NET in versione Open Source

Il progetto Mono è nato da un' iniziativa Ximian (oggi è appoggiato anche da Novell) con l'obiettivo di creare una versione Open Source per Unix della piattaforma di sviluppo Microsoft .NET, in maniera tale da permettere agli sviluppatori Unix di realizzare applicazioni .NET cross-plattform.

Mono include un compilatore per il linguaggio C#, un ambiente runtime chiamato Common Language Runtime (CLR) per il Common Language Infrastructure (CLI) e numerose librerie di classi completamente compatibili con .NET.

mono-1.0.1-gtksharp-1.0-win32-0.5.exe

Eclipse 3.0.1

Il massimo per lo sviluppo Java

Il progetto Eclipse, inizialmente di proprietà IBM e da questa donato alla comunità open-source è essenzialmente una raccolta di vari tipi di sottoprogetti che in realtà insieme formano, più che un software per lo sviluppo, un framework per la creazione di ambienti integrati di sviluppo per qualunque linguaggio e piattaforma.

Ma la ragione del successo di Eclipse risiede nell'IDE per la creazione di applicazioni Java che, offerto di fatto come esempio di uso del framework che Eclipse mette a disposizione, è riuscito a imporsi come prodotto di alta qualità, pur essendo completamente gratuito. L'installazione del prodotto è veramente molto semplice. Dovete assicurarvi che sia installato sulla vostra macchina un JDK pari o superiore al 1.4.1, dopodichè potete unzippare il file direttamente dove volete installare il prodotto… finito! Tra i vari file estratti troverete l'eseguibile eclipse.exe da lanciare per fare partire l'IDE.

eclipse-platform-3.0.1-win32.zip

mysql-4.1.5

Il più diffuso database server Open Source

MySQL è diventato in breve tempo il database server Open Source più popolare, con il tasso di crescita più elevato proprio all'interno delle industrie. Questo è dovuto alla volontà degli sviluppatori di fornire un prodotto efficiente e meno complicato dei rivali, che garantisce un TCO notevolmente ridotto. MySQL offre numerosi vantaggi: affidabilità e prestazioni elevate prima di ogni altra cosa, mentre la sicurezza è sempre di prim'ordine, grazie al lavoro di test svolto dalla comunità Open Source. Esistono numerose versioni del database server: Max DB, cluster e a breve una versione embedded. Inoltre, sono disponibili tutta una serie di strumenti che migliorano l'efficienza e la produttività:

- **MySQL Control Center** (MySQLCC) un client grafico per la gestione dei dati;
- **MySQL Administrator** un'interfaccia di amministrazione visuale per l'amministrazione del server MySQL;
- **MySQL Connector/J** e **MySQL Connector/ODBC** per la connessione attraverso driver JDBC (Java) e ODBC.

mysql-4.1.5-gamma-win.zip

DEVLIB 1.4

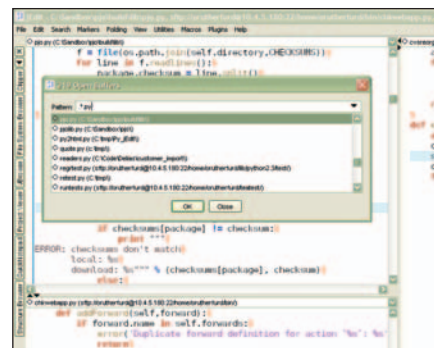
Il massimo per creare applicazioni multimediali.

Un framework object-oriented realizzato completamente in C++ con l'obiettivo di fornire agli sviluppatori un sofisticato strumento per la creazione di prodotti multimediali: giochi, screensaver, applicazioni grafiche (2D e 3D), player audio e video, ecc. DevLib è compatibile con Bloodshed C++ e Microsoft Visual C++.

DevLib-SDK-1.4.zip

jEdit 4.2

Un editor per programmatori Java

Realizzato completamente in Java, jEdit è un completo editor di testi per programmatori disponibile per numerose piattaforme tra cui MacOS X, OS/2, GNU/Linux (Unix) e Windows. Il programma, per l'immediatezza e la semplicità di utilizzo, è molto usato per scopi didattici e da programmatori non professionisti.

jEdit integra un completo linguaggio di macro e dispone di un'architettura facilmente estensibile mediante l'utilizzo di numerosi plugin facilmente gestibili tramite il "plugin manager". Inoltre, nonostante sia stato creato per sviluppare prevalentemente applicazioni Java, dispone di funzionalità per indentare ed evidenziare il codice per oltre ottanta linguaggi di programmazione. Per poter installare e utilizzare jEdit è necessario disporre del J2SDK 1.3 o 1.4

jedit42install.exe

Demo: come unire programmazione e arte

Nella Demoscene

C'era una volta la demo, esibizione di capacità tecniche e artistiche di talentuosi, spesso geniali, programmatori, grafici e musicisti. Oggi l'esistenza di questa élite è minacciata da Werkkzeug

Forse non tutti hanno avuto il piacere di ammirare gli effetti spettacolari che compongono una demo, dunque, prima di procedere, definiamo alcuni concetti fondamentali per la comprensione dell'articolo. Una demo è una presentazione multimediale non interattiva, caratterizzata da grafica 2D/3D generata in tempo reale, di solito sincronizzata con una colonna sonora. Immaginate un video, costituito da effetti speciali di forte impatto visivo, ottenuto rigorosamente via codice! L'insieme delle persone coinvolte nella creazione di demo e le loro attività, in particolare i demoparty, formano la Demoscene, in breve *Scene* o *Scena*.
A questo punto, in molti potrebbero chiedersi perché sprecare del tempo prezioso nello sviluppo di applicazioni apparentemente inutili. La risposta è semplice: oltre a rappresentare una forma d'arte che permette di fondere tecnologia e fantasia, la programmazione di demo è un hobby in grado di contribuire alla notorietà degli sviluppatori. Ogni produzione ha bisogno di diversi sceners, ovvero grafici, musicisti e programmatori organizzati in gruppi, anche se a volte la demo è frutto del genio di una singola persona. I vari demogroup si danno appuntamento in eventi chiamati party, solitamente tenuti nei paesi dell'Europa settentrionale, per presentare i propri capolavori ad una platea di colleghi e semplici curiosi. Potete pensare ad un demoparty come ad un mix tra una festa in cui scorrono fiumi di birra ed una serie di competizioni che prevedono premi per i progetti migliori.
Nell'introduzione, ho accennato ad una élite, difatti solo un ristretto numero di individui ha doti artistiche e conoscenze tecniche adeguate all'implementazione di una demo di successo. Basti pensare che tra i requisiti essenziali bisogna annoverare la completa padronanza di: un linguaggio di programmazione (i più usati sono C/C++, Assembly, Pascal, di rado Java) , una libreria grafica tra OpenGL e DirectX se si programma in ambienti Windows, concetti matematici avanzati…
I demogroup più influenti nella Demoscene rilasciano demo di qualità eccezionale anche grazie a programmi autoprodotti, gelosamente custoditi negli hard-disk più bui e polverosi. Per nostra fortuna, i membri del celebre gruppo Farbrausch hanno deciso di rendere pubblico *Werkkzeug*, il tool che ha consentito loro di proporre le produzioni più spettacolari degli ultimi anni. Finalmente i comuni mortali potranno smettere di chiedersi “ma come accidenti fanno a tirar fuori cose simili?!?” e cimentarsi nella programmazione di demo per magari figurare tra i vincitori di qualche contest internazionale!

LO STRUMENTO

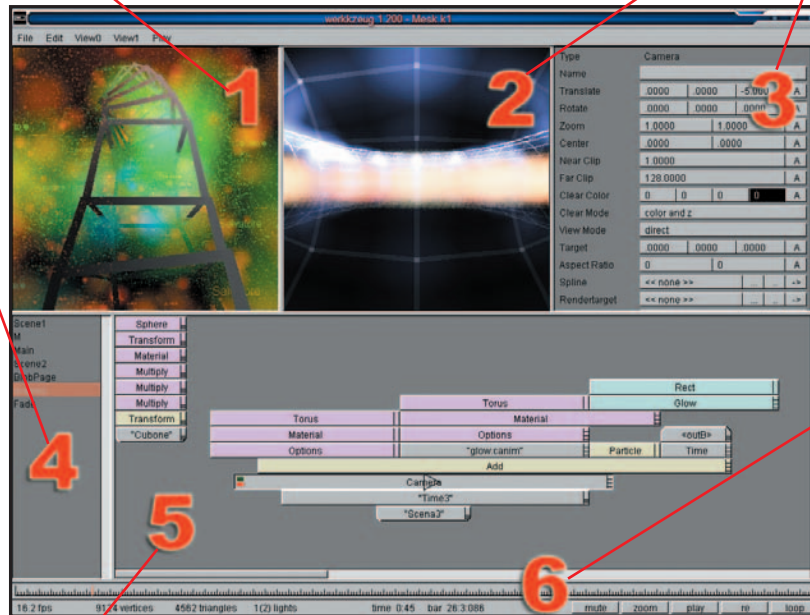
Werkkzeug è un programma molto controverso perché da una parte consente, anche al neofita, di ottenere in breve tempo effetti stupefacenti, dall'altra mina l'innovazione, lo spirito di gruppo della Demoscene e abbatte il mito romantico degli smanettoni in grado di sfruttare fino al limite le risorse hardware. Essendo poco interessati a questo genere di polemiche concentriamo la nostra attenzione sulle caratteristiche e sul funzionamento dello strumento in questione. I membri del gruppo Farbrausch considerano improponibile una documentazione completa del programma a causa della sua complessità.
Questo articolo, per ovvi motivi, non può trattare esaustivamente il prodotto, si propone quindi di fornire una descrizione introduttiva. Werkkzeug ha bisogno delle librerie DirectX 9.0a per operare correttamente, assicuratevi di aver aggiornato il vostro sistema prima di puntare l'indice contro ipotetici bug. Lo strumento può essere visto come un “linguaggio di programmazione visuale”, in stile Labview, che permette agli utenti di concentrarsi sulla creatività e sulla resa finale della demo. Tra le varie caratteristiche di Werkkzeug è d'obbligo sottolineare:

- Un motore 3D potente e veloce;
- Il supporto per svariati tipi di file: XSI (Softimage), LWO (modelli Lightwave), JPEG, XM (moduli musicali in formato eXtended Module), etc.

1. Questa finestra mostra l'anteprima in realtime dell'effetto o dell'intera demo. Per testare la resa di un aggregato di operatori è necessario selezionare un blocco e premere il tasto **s** (show). Potenza del "what you see is what you get"!

2. La seconda finestra di anteprima consente di esaminare un'altra porzione della demo al fine di eseguire un confronto per un eventuale perfezionamento dell'effetto. La combinazione di tasti che attiva l'anteprima secondaria è **ctrl+s**, non dimenticate però di selezionare l'operatore da visualizzare.

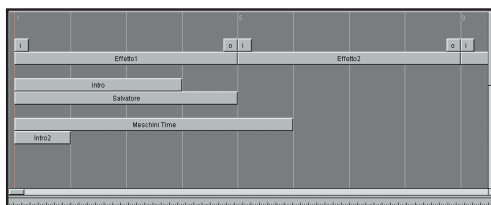
3. Ogni operatore è dotato di una serie di parametri, apportando delle modifiche ai valori di tali parametri si ottiene, in tempo reale, il risultato desiderato nella finestra di anteprima. Cliccando sulla *A* (animazione) posta vicino ad alcuni dei parametri si può stabilire il comportamento dinamico della proprietà. A titolo di esempio è possibile ruotare un oggetto attivando l'animazione del parametro rotate ed utilizzando un operatore "channel anim". Non vi preoccupate, tutto diventerà più chiaro in seguito!

4. La lista delle pagine serve per dare una struttura alla propria demo, avete presente i diversi moduli che compongono un'applicazione tradizionale? Bene, non si tratta di un concetto molto diverso. Le pagine contengono gli insiemi di operatori e dovrebbero essere create seguendo un criterio coerente. Se la demo è composta di tre scene è opportuno, ma non obbligatorio, definire tre pagine chiamate rispettivamente *Scena1*, *Scena2* e *Scena3* ed una pagina denominata *Main*. All'interno di una pagina il programmatore andrà a sovrapporre gli operatori creando così delle subroutine.

5. In questa area vanno disposti i vari blocchi, il colore del blocco ci aiuta ad individuare la classe di appartenenza dell'operatore. La lista degli operatori è accessibile direttamente per mezzo dei tasti che vanno da 0 a 4 (Fig. 3). L'ultima colonna delle liste è in comune alle varie classi. Con il pulsante destro del mouse si visualizza un menu contestuale ricco di voci. Potete modificare la visuale tenendo premuti il tasto b ed il pulsante destro del mouse mentre spostate il puntatore. Il tasto f invece apre una finestra di ricerca, se cliccate sul pulsante *Name* e selezionate l'opzione *Bugs* consulterete la lista degli errori potenziali.

6. In basso ci sono i controlli temporali quali la timeline ed i pulsanti per la riproduzione della demo. Non trascurate l'indicazione dei frame per secondo (FPS), al diminuire di tale valore corrisponde un rallentamento della demo. Una sottile linea rossa verticale evidenzia il tempo corrente della riproduzione, dopo aver premuto il pulsante *play* può essere tascinata liberamente. La riproduzione della demo è riavviabile tramite il pulsante etichettato re *(rewind)*.

Fig. 1: Riconosciamo le principali aree dell'interfaccia

PER SAPERNE DI PIÙ

Se avete voglia di fare un viaggio all'estero vi consiglio di non perdere i seguenti party; sui relativi siti trovate tutte le informazioni logistiche, i regolamenti ed ovviamente i progetti presentati nelle edizioni precedenti:

http://www.assembly.org
http://www.theparty.dk
http://www.takeover.nl
http://www.gathering.org
http://ms.demo.org

- Un generatore di texture e forme 3D;
- I controlli temporali e lo spline editor;
- L'elaborazione delle immagini, gli effetti speciali (FX) ed un potente sistema particellare;
- La gestione degli effetti tramite pixel shader;
- Un numero veramente notevole di operatori parametrizzabili;
- La presenza di un "compilatore" ottimizzato;
- Un ambiente di sviluppo completamente wysiwyg.

L'ambiente mette a disposizione un ampio numero di blocchi elementari, ma altamente configurabili, chiamati *operatori*. Tali operatori, graficamente, corrispondono a dei rettangoli colorati, sono paragonabili alle parole chiave di un linguaggio e devono essere combinati in modo opportuno per ottenere delle routine. Sovrapponendo gli operatori si definisce una connessione tra essi e di conseguenza la struttura della demo, vi faccio notare che le operazioni vengono eseguite dall'alto verso il basso ed una linea rossa evidenzia le sovrapposizioni incompatibili. Dopo aver analizzato l'interfaccia utente dell'ambiente Werkkzeug (Fig. 1) vedremo degli esempi concreti di routine create disponendo uno sull'altro diversi operatori. L'interfaccia è formata essenzialmente da sei aree a cui faremo spesso riferimento. L'interfaccia utente è formata anche da altre finestre, in particolare segnalo la visualizzazione temporale (Fig. 2) e lo spline editor (Fig. 4). Si noti che quasi tutte le funzioni del programma sono facilmente raggiungibili grazie ad un complesso di scorciatoie da tastiera. Se la finestra attiva è quella identificata dal numero 5 allora la pressione del tasto **e** mostra/nasconde la sequenza temporale della demo. Tale sequenza è modificabile trascinando i blocchi *time* oppure operando direttamente sui singoli parametri della zona 3. Quando due o più tempi coincidono il colore dei blocchi time incriminati passa da grigio a rosa. Lo spline editor risulta accessibile da ogni operatore di tipo *Camera* mediante il tasto **g**, si tratta di uno strumento potente ma purtroppo poco intuitivo. L'editor consente di stabilire i movimenti non-lineari della camera: i vostri primi esperimenti potranno tranquillamente farne a meno.

Fig. 2: Andamento temporale della demo. La visualizzazione è attivata/disattivata con il tasto e (premuto nell'area contenente gli operatori)

OPERATORI

Gli operatori sono classificabili in cinque categorie (Fig.3):

Fig. 3: Con i tasti 0-5 si accede alle diverse classi di operatori. Memorizzate le scorciatoie da tastiera!

- **Misc** – I blocchi strettamente necessari per implementare una demo in Werkkzeug sono: *Demo, Metronome* e *Time*. Vi accorgerete presto che non si può prescindere dalla conoscenza degli operatori *Camera, XM, Channel Anim, Load* e *Store*. Anche i commenti al codice, se così vogliamo definirlo, sono dei rettangoli colorati.
- **Bitmap** – Questi operatori sono utilissimi nella creazione di immagini bidimensionali utilizzabili come texture. *Glow, Pixels, Clouds, Blur* e *Add* sono tra i blocchi usati più comunemente.
- **Mesh** – Oggetti tridimensionali e relativi materiali trovano spazio nella categoria *mesh*. Complesse forme 3D vengono generate con procedure di estrusione, moltiplicazione, randomizzazione, trasformazioni generiche oppure operando su modelli creati con software di modellazione quali Softimage, Lightwave o l'economico Milkshape 3D.
- **Material** – Gli operatori *Material* e *Material Input* (classe *mesh*) definiscono le proprietà del materiale utilizzato nel rendering dell'oggetto.
- **Model** – *Particle* è un operatore molto interessante perché permette di simulare qualsiasi emettitore di particelle: fuochi d'artificio, fontane, nuvole, esplosioni, etc. Altri operatori da non trascurare sono: *Add, Transform, Font* e *Light*.

Werkkzeug viene distribuito con un'ottima guida, adatta sia come introduzione all'ambiente sia come riferimento per i programmatori, se così vogliamo definirli, esperti. Ogni categoria di operatori meriterebbe un articolo intero, dunque l'unico modo di imparare ad utilizzare il programma è partire dalla guida. Il file da caricare è chiamato *tutorial 133.k1*, se conoscete l'inglese vi consiglio vivamente di studiare il tutorial e di smanettare con gli operatori per prendere confidenza con il programma. Nella parte restante dell'articolo saranno messe in evidenza le funzionalità più rilevanti con un approccio hands-on.

LA PRIMA DEMO

In generale, il primo passo consiste nel creare un numero di pagine almeno uguale alle scene presenti nella demo da implementare. Per fare ciò dobbiamo premere il tasto **a** dopo esserci assicurati che l'area attiva sia la finestra della pagine. L'area *3* è dotata di un menu contestuale per la gestione delle pagine. Il nostro progetto, per non complicare troppo le cose, sarà contenuto in un'unica pagina in cui posizioneremo tutti gli operatori. Ogni blocco ha una linea verticale sulla destra, cliccandovi sopra e spostando il mouse il mouse verso destra o sinistra si ridimensiona l'oggetto. I blocchi possono essere spostati singolarmente o in gruppo previa una selezione simile a quella delle icone in Windows Explorer. Il programma offre anche la possibilità di tagliare, copiare e incollare gli operatori selezionati, premendo **x**, **c**, **v** (senza *CTRL*). Poniamoci adesso l'obiettivo di creare una scena che mostri una struttura 3D in movimento all'interno di una nebulosa (Fig.1):

Fig. 4: Lo spline editor, uno strumento potente ma poco intuitivo. È accessibile con il tasto g da ogni operatore Camera

- Abbiamo bisogno di diversi operatori della classe *Bitmap* per generare un materiale. Disponiamo i blocchi Pixels (con Count = 4096), *Glow, Add, Text* e *Add* come in (Fig.5) e aggiungiamo un operatore *Material* chiamandolo *Materiale*.
- L'oggetto 3D viene creato partendo da un banalissimo cubo attraverso un procedimento di estrusione con *Count = 5* ed un minimo di rotazione. Il modello appare vuoto perché l'operatore *Options* ha il parametro *Generate* posto a *"Thick Lines"* ovvero *"Linee Spesse"*.
- L'effetto nebulosa è dovuto a due blocchi *Particle* con la proprietà *Sprite Material* impostata al materiale precedentemente generato. I due sistemi particellari rappresentati dagli operatori *Particle* funzionano in parallelo perché sono combinati entrambi con l'oggetto 3D per mezzo del blocco *Add*. Modificando leggermente i diversi parametri dell'operatore *Particle* si ottengono effetti molto interessanti.
- Il movimento di camera è gestito da due trasformazioni distinte: una traslazione ed una rotazione. Il blocco *Camera* offre un'ampia gamma di parametri, dalle distanze di clipping alla modalità di vista passando per le spline e l'integrazione con gli effetti speciali (operatore *FX*). Il programma può funzionare anche in modalità "modifica diretta": selezionate una delle due anteprime e nel relativo menu *ViewX* spuntate la voce *Link Edit*. Tutte le modifiche apportate all'anteprima si rifletteranno sui parametri dell'oggetto.
- Werkkzeug implementa

Fig. 5: Il "sorgente" della nostra prima demo

SUL WEB

Se avete bisogno di risorse multimediali quali moduli musicali o modelli 3D seguite i link:

www.modarchive.com
www.dc5.org/demodulate
Due siti da cui è possibile scaricare moduli musicali, alcuni file sono coperti da diritti d'autore.

www.modplug.com/modplug
Provate questo tracker per mettere alla prova il musicista che è in voi.

www.swissquake.ch/chumbalum-soft
Modellatore 3D, vi ricordo che .Werkkzeug1 supporta i formati XSI e LWO.

il concetto di subroutine con gli operatori *Load* e *Store*, gli unici caratterizzati da bordi arrotondati. Ad intere sequenze di operatori viene assegnato un nome dal blocco *Store*, la chiamata della subroutine è indipendente dalla pagina in cui essa è definita e avviene mediante l'operatore *Load*. Nel nostro esempio l'animazione risulta immagazzinata in *"IoPScena"* ed è richiamata dal blocco *Load* denominato *"MostraScena"* con un apposito link.

• La demo è quasi pronta, non resta che definirne l'andamento temporale. Di solito conviene associare un operatore *time* ad una scena per fissare l'istante iniziale e la sua durata. Ogni demo deve necessariamente contenere un operatore di tipo demo chiamato *"root"*, ossia radice, visualizzabile con il tasto **r**. A tale operatore vanno collegati, sempre con la solita disposizione a cascata, un metronomo e la scena salvata in precedenza. Lo strumento permette di importare due formati musicali distinti: *XM* e *OGG*. Si tratta rispettivamente di moduli composti via tracker, con l'utilizzo di suoni campionati, e di file audio tipo MP3. Se volete demo di dimensioni ridotte puntate sui moduli *XM* composti da voi oppure scaricati da Internet.

• Finalmente possiamo compilare il nostro "capolavoro", generando così un eseguibile. Per fare ciò dobbiamo scegliere l'opzione *Make* dal menu *File*. A questo punto ci verrà chiesto se creare una demo, una intro oppure un programma custom. Una *intro* rappresenta la versione limitata, per quanto riguarda gli operatori impiegabili, della *demo*. Il vantaggio delle intro rispetto alle demo sta tutto nella dimensione dell'eseguibile, tendenzialmente inferiore ai 64kbyte! L'opzione *custom* purtroppo non è disponibile pubblicamente.

Un metodo alternativo per animare gli oggetti, e più in generale i parametri dotati dell'attributo *A*, consiste nell'inserire un operatore di tipo *Channel Anim* (Fig. 6).

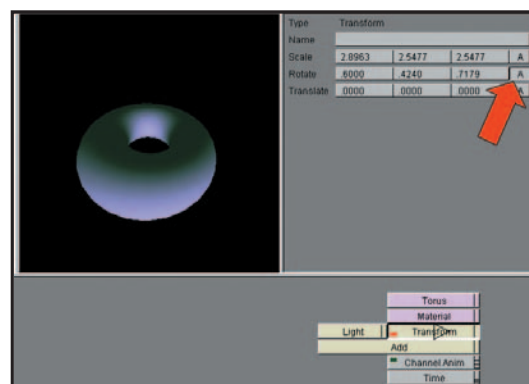



Fig. 6: Animazione con l'operatore Channel Anim

Tutti i parametri che condividono lo stesso valore di channel saranno animati in modalità single o double, è anche possibile specificare la modulazione:

Fig. 7: Tutti i parametri sono fondamentali! Animate deve essere impostato su ON

rect, a rampa positiva o negativa, sinusoidale, triangolare, etc. Cliccando sulla *A* appare un pannello (Fig. 7) per la configurazione dell'animazione in cui dovete porre *Animate* su *ON*, fate attenzione nella scelta dei valori perché sono tutti importanti! A questo punto selezionate l'operatore *Channel Anim*, premete **s** e successivamente il pulsante *play*.
Se l'effetto non vi soddisfa probabilmente è il caso di aumentare la sua intensità tramite il parametro *Amplify*. Abbiamo usato solo una minima parte degli operatori presenti in Werkkzeug, considerando che i vari blocchi possono essere sovrapposti in tantissimi modi diversi e personalizzati con una miriade di parametri vi lascio immaginare la flessibilità dello strumento. I progetti salvati da Werkkzeug hanno estensione *.k1*, da non confondere con i file *.kk1*. Un file *.kk1* è il risultato dell'esportazione della demo con il comando File/Export e può essere riprodotto con l'ausilio del player *kkino1.exe*. Vi faccio notare che l'eseguibile prodotto da *File->Make* non è altro che il player unito alle procedure e alle risorse presenti nel *.k1*. Al momento sono disponibili due file *.k1*, creati dal gruppo Farbrausch, da cui imparare i trucchi del mestiere: il tutorial ed il "sorgente" di *FR-025-The.Popular.Demo*. Il CD allegato alla rivista contiene il file *.k1* relativo alla demo proposta nell'articolo ed una piccola sorpresa!

CONCLUSIONI

Questo articolo è stato scritto a pochi giorni di distanza dal rilascio del programma, confido nell'interesse da parte degli sviluppatori e di conseguenza nell'aumento delle risorse ad esso dedicate.
Spero di aver prodotto in voi, nonostante il limitato spazio a mia disposizione, uno stimolo ad approfondire l'argomento. Lo strumento proposto, unito alla "rinomata italica creatività", potrebbe contribuire alla realizzazione di demo di successo. Al fine di promuovere l'utilizzo di Werkkzeug ho organizzato un piccolo contest, visitate il sito *http://smeschini.altervista.org* per avere maggiori informazioni sulla competizione! I membri del gruppo Farbrausch stanno lavorando alla terza versione di Werkkzeug, tenete sottocchio gli sviluppi del progetto.
Alla prossima!

Salvatore Meschini

www.theprodukkt.com
È il sito ufficiale del "prodotto".

www.scene.org
Un punto di riferimento per la Demoscene: novità, demo, eventi...

http://smeschini.altervista.org
Guide, esempi ed altro materiale interessante.

www.vectronix.org/wbb
Un forum con trucchi, tutorial ed esempi.

Una midlet per inviare i nostri sms "quasi" gratuitamente

SMS gratis!

Scriviamo una applicazione per inviare SMS a meno di un centesimo di euro. Impareremo a far comunicare una midlet con una servlet e scopriremo come ridurre i byte scambiati via GPRS

Questo mese vedremo come sia possibile far interagire una nostra midlet, installata su un terminale mobile GPRS (probabilmente un semplice telefonino!), con una servlet scritta "ad hoc". Ma il fine ultimo è molto meno nobile: al di là degli "scopi didattici", sfrutteremo questa interazione per utilizzare un servizio offerto da Vodafone Italia a tutti gli utenti iscritti al suo portale 190.it. Il servizio in questione si chiama *"SMS via mail"*, titolo abbastanza esplicativo che dovrebbe farvi intuire quale sarà l'utilizzo che faremo dell'applicazione finita.

IL SERVIZIO UTILIZZATO

"Sms via e-mail" è un servizio offerto da Vodafone Italia a tutti i suoi clienti, se iscritti al portale 190.it, che permette di inviare fino a 10 SMS al giorno verso numeri Vodafone Italia in maniera completamente gratuita. L'utilizzo del servizio è abbastanza semplice: inviando una mail all'indirizzo *numero_destinatario@sms.vodafone.it*, verrà inviato un messaggio SMS al numero fornito, con il testo del subject e del body della mail. La mail deve essere inviata indicando come mittente l'indirizzo e-mail fornito in fase di registrazione al sito 190.it, lo stesso su cui avete ricevuto la conferma di registrazione e al quale Vodafone invia la sua NewsLetter periodica. Dovrete rispettare una sola altra limitazione: la lunghezza del messaggio non deve essere superiore a 120 caratteri circa (il numero massimo di caratteri varia a seconda dell'indirizzo mail del mittente, che ovviamente Vodafone include nel SMS per evitare messaggi anonimi). Come dicevo, il servizio è completamente gratuito, ad eccezione del costo sostenuto per l'invio della e-mail, che nel nostro caso è calcolato in base al traffico GPRS generato dall'applicazione. La tariffazione del collegamento GPRS non è, infatti, dipendente dal tempo di connessione, ma solo dal numero di KB scambiati; utilizzando una sim Vodafone, spenderemo 0,06 euro per ogni KB scambiato. Ricordate che potete utilizzare l'applicazione attraverso qualsiasi operatore (anche Tim e Wind), l'essenziale è che siate registrati in 190.it. Fra le linee guida da rispettare vogliamo quindi tenere conto del traffico generato, in modo da minimizzare i costi sostenuti. Vedrete che, ad applicazione finita, riusciremo a inviare un messaggio con un costo quasi nullo. Costo che si azzera del tutto per chi utilizza tariffe GPRS FLAT, cioè tariffe che sono indipendenti dal tempo di connessione e dalla quantità di byte scambiata.

IL PROGETTO

Le nostre scelte implementative dovranno dipendere, oltre che dalla minimizzazione del traffico generato, anche dalla ricerca della massima compatibilità con i vari terminali presenti sul mercato e con le configurazioni di accesso GPRS dei vari operatori mobili. In particolare, l'applicazione deve funzionare sia con un profilo GPRS di tipo *"web"* sia con un profilo *"wap"*. Questo perchè spesso i gestori offrono tariffe "a forfait" per l'accesso al wap via GPRS e noi non vogliamo scartare la possibilità di usufruirne con la nostra applicazione. Quindi, la connessione diretta ad un server SMTP via Socket andrebbe scartata, visto che in connessione wap non è ovviamente attuabile. La connessione diretta via SMTP va esclusa anche per la ricerca della compatibilità con i terminali, visto che la *SocketConnection* spesso non è supportata dai terminali MIDP 1.0, anche se secondo le API dovrebbe esserlo. Infine, non sceglieremo la connessione diretta per minimizzare il traffico generato. Infatti, una sessione SMTP per l'invio di mail richiede una serie di comandi e di autenticazioni (standardizzati nel RFC 821) che la renderebbero ben più esosa, sia in invio che in ricezione, di una semplice richiesta ad una servlet, come vediamo nell'esempio:

```
server> 220 BBN-UNIX.ARPA Simple Mail Transfer
                                 Service Ready
client> HELO mail.tin.it
server> 250 fep02-svc.tin.it
```

NOTA

190.it è il portale di Vodafone Italia che offre agli utenti iscritti, oltre all'utilissimo "sms via e-mail", la possibilità di inviare 100 sms al giorno dal web, MMS gratuiti e, ovviamente, la gestione della propria sim Vodafone.

```
client> MAIL FROM:
server> 250 Sender   OK
client> RCPT TO:
server> 250 Recipient   OK
client> DATA
server> 354 OK Send data ending with .
client> From: xxx@javastaff.com
client> To: info@javastaff.com
client> Subject: prova
client> prova di trasmissione.
client> Sto provando il protocollo SMTP
client> [Resto del messaggio]
client> .
server> 250 Message received
client> QUIT
server> 221 fep02-svc.tin.it ESMTP server closing connection
```

Mentre una semplice richiesta HTTP GET è più "leggera":

```
client>GET http://nomeserver/apps/smssender.php
                                        HTTP/1.0
server>HTTP/1.1 200 OK
server>Date: Wed, 17 Sep 2004 08:18:47 GMT
server>Server: Apache/1.3.26 (Unix) PHP/4.2.3
server>Expires: Thu, 19 Nov 1981 08:52:00 GMT
server>Cache-Control: no-store, no-cache, post-check
                                  =0, pre-check=0
server>Connection: close
server>Content-Type: text/html
server>[Contenuto della pagina HTML..]
```

Dovremo ricordarci quindi di far restituire alla servlet una risposta breve, per non sprecare inutili byte. Bisogna però analizzare se può una semplice (e quindi leggera) richiesta GET essere sufficiente per i nostri scopi o se dobbiamo scegliere una POST, che solitamente genera più traffico. La limitazione di una richiesta di tipo GET, infatti, è sulla lunghezza dell'URL, che secondo lo standard http (RFC 2616) è di massimo 256 caratteri. Possono bastarci? Facciamo un rapido calcolo: per il messaggio ci servono massimo 120 caratteri, la lunghezza di un indirizzo mail è mediamente inferiore a 25/30 caratteri (es: *luca.mattei@javastaff.com*), un numero di cellulare è lungo 10 caratteri. Ne rimangono quasi 100 per indicare il path della servlet e per i nomi dei parametri: sono più che sufficienti! Quindi, quella che sembrava una limitazione (i 256 caratteri massimi della URL) si rivela un utile aiuto anche nell'ottica della ricerca della minimizzazione dei byte scambiati: in uscita avremo massimo 300 caratteri (e quindi byte). Confermiamo come valida la scelta di utilizzare una servlet come tramite per trasmettere la mail al server SMTP che si prenderà carico di consegnarla al server della Vodafone. Possiamo quindi pensare ad una infrastruttura del tipo rappresentata in Fig. 1.

Fig. 1: ***Una rappresentazione della infrastruttura con le interazioni fra i diversi "agenti" protagonisti della transazione di un messaggio***

J2ME (Java 2 Mobile Edition) è la versione di Java 2 Standard Edition J2SE, ottimizzata per girare su cellulari e Palmari di ultima generazione. Una MIDlet è quindi una applicazione J2ME.

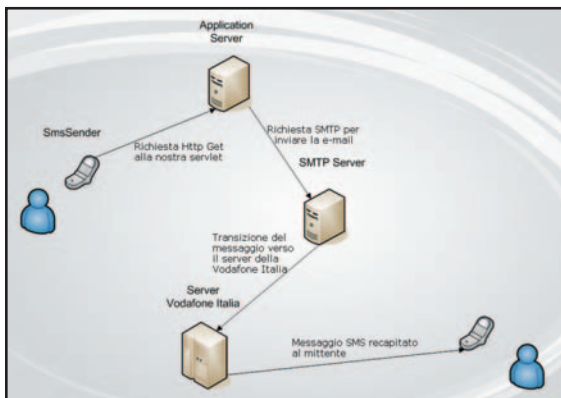
LA SERVLET

Iniziamo analizzando la parte server del progetto, che è una semplice Servlet con cui la nostra midlet dovrà dialogare. In sostanza, la servlet deve essere in grado di ricevere alcuni parametri attraverso la richiesta GET e, dopo averli elaborati, connettersi verso un server SMTP e inviare la mail all'indirizzo *"numero_destinatario@sms.vodafone.it"*. La servlet quindi sarà piuttosto semplice, dovendo implementare solo il metodo *doGet* (possiamo evitare il *doPost*, dato che abbiamo deciso di non utilizzare la Post Http). I parametri della *Get Http* sono disponibili attraverso il metodo: *request.getParameter("nome_parametro")* dove *request* è una *HttpServletRequest*, che ci viene passata nel metodo *doGet*. I parametri di cui abbiamo bisogno sono il numero del destinatario, il messaggio vero e proprio e la mail del mittente:

```
String to = request.getParameter("to");
String from = request.getParameter("from");
String text = request.getParameter("text");
```

Attraverso la *HttpServletResponse*, possiamo comunicare al client l'esito dell'operazione. Basta utilizzare il *PrintWriter* ottenuto attraverso il metodo *getWriter()*. Vediamo in dettaglio il codice che gestisce il dialogo con il client:

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response) throws
                       ServletException, IOException {
  PrintWriter out;
  String to = request.getParameter("to");
  String from = request.getParameter("from");
  String text = request.getParameter("text");
  response.setContentType("text/html");
  out = response.getWriter();
  try{
    SendMail(to+"@sms.vodafone.it",from,text);
    out.println("Messaggio Inviato\n");}
  catch(Exception e) {
    out.println("Errore in invio:\n"+e.toString());}
  out.close(); }
```

Resta da vedere come implementare il metodo: *SendMail(String destinatario,String mittente, String*

testo). Potremmo utilizzare il package *JavaMail*, ma preferiamo vedere in dettaglio come sia possibile inviare una mail utilizzando una Socket per dialogare con un server SMTP. Secondo l'RFC 821, che è lo standard per la comunicazione via SMTP, il client deve inviare in sequenza una serie di comandi ai quali, volta per volta il server risponde con un diverso messaggio di successo. Ad esempio al comando:

```
HELO nomeserver.com
```

il server risponde, in caso di successo, con:

```
250 fep02-svc.tin.it
```

In cui 250 è il codice di successo. Analizziamo ora il metodo che ci permette di inviare la mail:

```
SendMail(String to, String from, String message)
                    throws ProtocolException, IOException{
  Socket socket = new Socket(MAIL_SERVER_URL,
                                         MAIL_PORT);
  in = new DataInputStream(socket.getInputStream());
  out = new PrintStream(socket.getOutputStream());
  if(!checkStatus("220")) throw new ProtocolException(str);
    out.println( "HELO " + MAIL_SERVER_URL );
    out.flush() ;
  if(!checkStatus("250")) throw new ProtocolException(str);
    out.println( "MAIL FROM: " + from );
    out.flush() ;
  if(!checkStatus("250")) throw new ProtocolException(str);
     out.println( "RCPT TO: " + to );
     out.flush() ;
  if(!checkStatus("250")) throw new ProtocolException(str);
     out.println( "DATA" );
     out.flush() ;
  if(!checkStatus("354")) throw new ProtocolException(str);
     /* Il codice di successo questa volta non è 250,
                                          ma 354; *
    * da adesso il client trasmette la mail vera e
                            propria, conclusa da   *
    * un punto su una linea vuota. */
    out.println("From: " + from);
    out.println("To: " + to);
    out.println( "Subject: \n" ); //il Subject non è
                         necessario, lo lasciamo vuoto
    out.flush() ;
    out.println("X-Mailer: SmsSender Servlet 1.0");
    out.println("");
    out.println( message ) ;
    out.println(".") ;
    out.flush() ;
  if(!checkStatus("250")) throw new ProtocolException(str);
    out.println("QUIT");
    out.flush();
    in.close();
   socket.close();
}
```

Avrete notato che abbiamo utilizzato il metodo *checkStatus(String codice)* per verificare che il server avesse "capito" la nostra richiesta. Questo metodo non fa altro che ricevere dalla *DataInputStream* la risposta e verificare se i primi caratteri corrispondono al codice che ci attendiamo di ricevere. Se qualcosa è andato storto, lanceremo una eccezione:

```
private boolean checkStatus(String RFC)throws
                          ProtocolException,IOException{
  str = in.readLine();
  System.out.println(str);
  if (!str.startsWith(RFC)) throw new ProtocolException(str);
     while (str.indexOf('-') == 3) {
       str = in.readLine();
    if (!str.startsWith("RFC"))  throw new
                                   ProtocolException(str);
  }
  return true;
}
```

Potete installare la vostra servlet, magari con qualche modifica, su *MyJavaServer.com*, un server che offre hosting java gratuitamente. Il processo di compilazione e installazione è abbastanza semplice, dato che avviene dal pannello di controllo del sito con una semplice form. Basta procedere all'upload del file *.java* via ftp, quindi, una volta identificati in *"member area"*, cliccare su *"java compiler"* e indicare il nome del file. Se per caso non avete voglia di creare un vostro account, non disperate, potete utilizzare quella appositamente utilizzata per scrivere questo articolo, che è reperibile all'URL: *www.myjavaserver.com/servlet/pdfcrawler.SmsSenderServlet.* Ora manca solo un client in grado di utilizzarla.

GLOSSARIO

J2EE (Java 2 Enterprise Edition), è la parte di java che la Sun ha dedicato alle applicazioni server. Una servlet è una applicazione java fatta per girare su un application server e può essere interrogata via http.

QUANTO SI RISPARMIA?

Il costo del singolo messaggio inviato con SMSSender dipende sia dalla tariffa che dal protocollo utilizzato per il collegamento. Per coloro i quali utilizzano il protocollo GPRS, la dimensione del singolo messaggio (comprensiva di tutti i dati scambiati con il server) sarà di circa 1KB, mentre per chi utilizza un collegamento Wap, la dimensione media si aggirerà sui 2,5KB. Ovviamente, per chi gode di una tariffa flat, il costo del singolo messaggio si azzera.

	Via GPRS	Via WAP
TIM	0,6	4
Vodafone	0,6	3
Wind	0,3	4
H3G - 3	da 0,2 a 0,6 a seconda dei piani tariffari	non disponibile

I valori sono espressi in centesimi di euro per KByte.

Per calcolare il costo del messaggio, basta moltiplicare la dimensione in byte per il coefficiente indicato in tabella. Ad esempio, un SMS inviato con scheda Vodafone via GPRS costerà (circa) 1[KB] * 0,6 [eurocent/KB] = 0,6 centesimi di Euro. Il che vuol dire che, al costo di un Euro, potremo mandare oltre 160 messaggi!

URL ENCODING

Una premessa è necessaria, prima di intraprendere la scrittura della midlet client. Lo standard HTTP prevede che in una richiesta *Get*, i parametri vengano accodati all'indirizzo, dopo un *"?"* divisi dal carattere *"&"*. Ad esempio, se la variabile da passare è *text*, il cui valore è *"prova"*, avremo: *http://server/servlets /SendMail?text=prova&destinatario=3401234567*. I valori delle variabili vanno però codificati seguendo alcune regole. Bisogna infatti sostituire alcuni caratteri non ammessi con il loro equivalente UNICODE. Questi caratteri sono solitamente caratteri riservati, per esempio il simbolo di "dollaro" *("$")* quello di "e commerciale" *("&")*, lo spazio e così via. Nella *Tab. 1* trovate alcune codifiche UNICODE corrispondenti ai più comuni caratteri "riservati" che possono essere utilizzati in un SMS in lingua italiana. Dovremo quindi scrivere un metodo che, data in ingresso una stringa contenente l'URL per la GET, ne restituisca un'altra con i caratteri codificati. Purtroppo un metodo del genere in MIDP 1.0 non esiste. Ma non ci perdiamo d'animo: ne scriviamo uno ad hoc!

Fig. 2: Il form per la compilazione delle servlet su MyJavaServer

```
public String urlEncode(String url){
  url=replace(url,'à',"%E0");
  url=replace(url,'è',"%E8");
  ...
  return url;}
public String replace(String origine, char orig, String dest){
  String appoggio=origine;
  for (int i=0;i<appoggio.length();i++)
    if (appoggio.charAt(i)!=orig)
      origine+=appoggio.charAt(i);
   else origine+=dest;
  return origine; }
```

MyJavaServer.com è un application server gratuito e molto funzionale, che ci dà la possibilità di mettere online le nostre applicazioni java server. Supporta: J2SE 1.5, Servlet 2.3, Java Server Pages (JSP) 1.2

à	=	%E0
è	=	%E8
é	=	%E9
ì	=	%EC
ò	=	%F2
ù	=	%F9
$	=	%24
#	=	%23
£	=	%A3
@	=	%40
'	=	%27
spazio	=	%20

Tabella 1: I più utilizzati codici UNICODE

LA RICHIESTA GET

Incominciamo la scrittura della midlet partendo dalla parte più importante: l'invio del messaggio attraverso una richiesta *"Http Get"*. Facciamo conto di aver già ottenuto dall'interfaccia grafica il messaggio e il numero del destinatario. La mail del mittente possiamo definirla direttamente nel codice, scelta veloce, ma valida solo se si vuole essere gli unici utilizzatori della midlet, con la riscrittura della stessa ogni volta che un amico o collega ce la chiederà… e fidatevi che vi accadrà facilmente. Inoltre potrete utilizzare un solo account, mentre, se avete più di una sim Vodafone, potreste inviare più di 10 SMS al giorno, con una gestione della mail mittente più accurata. Quindi, un'idea potrebbe essere quella di una form per l'immissione dell'indirizzo, con conseguente salvataggio su RMS. Ma partiamo dalla base, la richiesta GET: il package *javax.microedition.io* ci mette a disposizione la classe *HttpConnection* dalla quale possiamo ottenere l'InputStream su cui leggere la risposta. La richiesta vera e propria la farà java, una volta aperto il Connector attraverso il metodo *open(String url)*. Ma vediamo direttamente il codice:

```
URL=URLServlet + "?text=" + mess +"&to=" +
                              n+"&from="+MailMittente;
URL=urlEncode(URL);
String response = "";
HttpConnection c = null;
try {
  // apriamo la connessione http all'indirizzo specficato sopra
  c = (HttpConnection)Connector.open(URL);
  // una volta che la connessione è stabilita
  // apre un input stream
  InputStream is = c.openInputStream();
  int ch;
  // legge byte per byte l'inputStream e 'appende'
  // i singoli caratteri nel buffer response
  while ((ch = is.read()) != -1) {
    response=response+( (char)ch );}
  // chiudiamo l'InputStream.
  is.close();
  // chiudiamo la connessione
  c.close(); }
catch ( IOException e ) {
  //notifica l'eccezione }
```

Se la servlet ha funzionato a dovere la stringa buffer conterrà il risultato *"Messaggio Inviato"*, che potremo notificare direttamente all'utente. Da notare la composizione dell'URL, che incorpora, come da specifiche dell'HTTP, il messaggio, il mittente e il destinatario, divisi dal carattere *"&"*:

```
URL=URLServlet + "?text=" + mess +"&to=" +
                              n+"&from="+MailMittente;
```

Per quanto riguarda la segnalazione del mancato invio del messaggio, può avvenire per un errore sulla servlet o nella comunicazione fra la midlet e la servlet. I possibili errori della servlet sono automaticamente mostrati nella risposta che troviamo in *response*. Infatti, come ricorderete, nel catch della servlet avevamo fatto in modo che fosse riportata chiesto la motivazione dell'eccezione lanciata:

```
out.println("Errore in invio:\n"+e.toString());
```

La stessa cosa possiamo farla in caso di errori nel tentativo di fare la richiesta GET. Quindi, nel *catch* del metodo *Send* della midlet, potremo accodare a

response il motivo dell'eccezione lanciata, in modo che nell'interfaccia grafica non dovremo fare altro che scrivere la stringa *"Messaggio Inviato"* nella Form finale, che corrisponderà al successo dell'operazione. Il nucleo dell'applicazione è questo, niente di difficile, e con il vantaggio di mantenere i dati inviati e ricevuti sotto una soglia considerevole e soprattutto di poter essere effettuata praticamente da tutti i cellulari, anche da quelli più "anziani". Il resto altro non è che un'interfaccia grafica per l'inserimento del messaggio e del numero del destinatario ed eventualmente per la gestione mittente utilizzato.

L'INTERFACCIA UTENTE

Anche nella scrittura dell'interfaccia grafica dovremo mirare ad un obiettivo: l'applicazione dovrà essere il più veloce possibile, evidentemente perché speriamo che venga utilizzata molto di frequente.
È ovvio che, se risultasse macchinosa o lenta, tenderemmo a preferire l'uso del composer standard del telefono, anche se più costoso. In questo purtroppo siamo ostacolati da un problema insormontabile: l'impossibilità di accedere alla rubrica. Per la verità, su alcuni Siemens il mittente può essere preso da rubrica. Il produttore ha pensato bene di poter far scegliere, nel caso di un *TextField* costruito con *TextField.PHONENUMBER* come parametro, il numero dalla rubrica interna. Va sottolineato che l'impossibilità di accedere alla rubrica attraverso J2ME discende da una ben precisa scelta dei progettisti in base a criteri di sicurezza. Una midlet, come una applet, viene eseguita in una *"SandBox"* che la isola dal resto del sistema operativo del terminale, evitando l'accesso a dati che potrebbero essere privati. D'altra parte, però, si limitano fortemente gli sviluppatori nel loro lavoro rendendo solitamente le applicazioni J2ME povere di funzionalità, soprattutto se confrontate ad applicativi per smartphone, scritti nel linguaggio nativo del sistema operativo (per esempio Symbian), che possono accedere a tutte le funzionalità del cellulare, senza limitazioni. Capirete quindi che, benché l'applicazione sia progettata per poter girare su quanti più terminali possibili, su alcuni, grazie all'impegno della casa madre, avremo delle funzionalità aggiunte che non ci aspettavamo ma che ci fanno indubbiamente piacere. Comunque, per non rendere macchinoso l'utilizzo della midlet, eviteremo inutili schermate di riepilogo o di conferma, cosa che renderà forse spartana la nostra applicazione, ma ne salvaguarderemo la funzionalità e la rapidità. Ciononostante, potremo permetterci una piccola immagine, come il logo del programma, nella schermata principale, a patto che l'operazione di caricamento e visualizzazione della *png* siano racchiuse in costrutti di tipo *try* e *catch* che non impediscano l'esecuzione su terminali che non le supportano (ad esempio perché dotati di display monocromatico). L'applicazione potrà, quindi, essere strutturata in tre diverse *Form*, una iniziale, con due *TextField* (numero destinatario e messaggio), una di presentazione del risultato di invio e una per l'immissione e la modifica della mail mittente (Fig. 3). Analizziamo la prima Schermata. Innanzitutto avremo istanziato la Form, che rappresenta lo "Screen" su cui andare ad "appendere" i vari oggetti grafici *("Field")* da utilizzare. Avremo quindi una Form *aForm* su cui "appendere" i vari *TextField* e i *Command* che ci permettono di passare agli stadi successivi della applicazione. Fondamentale è il controllo su *MailMittente,* ottenuta attraverso il metodo *readRecords()* che va ad accedere via RMS alla mail salvata e se non è presente, fa partire automaticamente la form per l'immissione e il salvataggio:

```
public void Setup() {
  inviando = new Form("Inserire mail");
  salvataggio = new Command("Invia", Command.OK, 1);
  exit = new Command("Esci", Command.EXIT, 2);
  String istruzioni="Inserire indirizzo indicato in fase di
                        registrazione sul sito www.190.it";
  mail=new TextField("Mail:", "", 120,  TextField.EMAILADDR);
  inviando.append(istruzioni);
  inviando.append(mail);
  inviando.addCommand(salvataggio);
  inviando.addCommand(exit);
  inviando.setCommandListener(this);
  d.setCurrent(inviando); }
```

Il *Command salvataggio* avvierà una funzione per salvare la mail su *RecordStore (RMS)*. Per salvare e caricare una stringa, scriviamo due funzioni, che utilizzano i metodi *addRecord(rec, 0, rec.length)* e *getRecord(i, recData, 0)* che ci fornisce il dato precedentemente salvato in *recData,* un array di byte. Passare da un array di byte a una stringa è facile, attraverso il costruttore *new String(byte[])*. L'operazione inversa avviene attraverso il metodo *String.getBytes(),* che restituisce proprio un *byte[]*. La terza e ultima Form dovrà solo presentare i risultati e permettere il ritorno alla schermata principale, quindi basterà integrare nella funzione per l'invio dei dati via Http Get, tutti i *Command* e l'append della stringa *result:*

```
inviando = new Form("Invio in corso");
torna = new Command("Ritorna", Command.OK, 1);
exit = new Command("Esci", Command.EXIT, 2);
inviando.addCommand(exit);
inviando.setCommandListener(this);
inviando.append("Connessione in corso..");
d.setCurrent(inviando);
//istruzioni per l'invio del messaggio come visto prima
inviando.append(response);
```

***Fig. 3:* Le tre Form utilizzate dalla nostra applicazione. Nella seconda screenshot la conferma che il messaggio è stato inviato**

```
inviando.addCommand(torna);
```

Probabilmente vi starete chiedendo chi gestisce il passaggio da una Form ad un'altra. La nostra midlet, per poter utilizzare i *Command*, deve implementare una interfaccia *(CommandListener)* ed in particolare un metodo: *commandAction(Command c, Displayable s)*. Qui basterà riconoscere quale è l'azione richiesta e richiamare il metodo voluto.
Per esempio:

```
if (c = = invia){send(); }
```

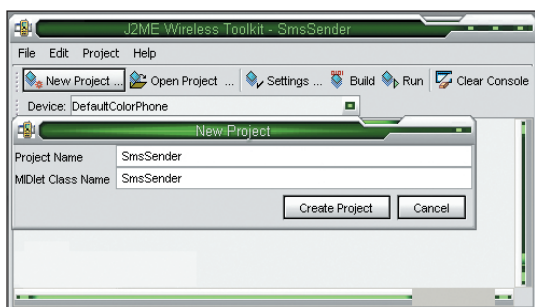



Fig. 4: Diamo il nome alla nostra nuova "creatura"

Tutto ciò risulta valido se *invia* è un *Command* definito in maniera globale nella nostra midlet, come abbiamo fatto nel nostro progetto. Se invece preferiamo definire i *Command* di volta in volta in ogni metodo, il riconoscimento possiamo farlo sulla label assegnata:

```
//se invia è così definito:
//invia = new Command("Invia", Command.OK, 1);
//possiamo usare:
if (c.getLabel().equals("Invia")){
   send();
}
```

Fig. 5: Ecco i primi passi della nostra MIDlet sull'emulatore

CREARE L'APPLICAZIONE

Abbiamo visto come disegnare le tre Form e come gestire il passaggio da una all'altra, ora vediamo come mettere insieme tutti i pezzi ed in particolare come creare l'archivio da installare sul cellulare. Una midlet, per poter essere riconosciuta tale, deve estendere la classe *javax.microedition.midlet.MIDlet* e implementare i tre metodi *startApp, pauseApp* e *destroyApp*. I tre metodi vengono richiamati dal cellulare rispettivamente quando l'applicazione viene avviata, messa in pausa (per l'arrivo di una chiamata per esempio) e chiusa. Nel nostro caso *startApp* avvierà il metodo che visualizza la prima Form, *pauseApp* invece, essendo la nostra applicazione del tutto sequenziale ed interattiva, non dovrà fare nulla. Lo stesso discorso vale per *destroyApp* dato che non abbiamo dati da salvare o altre operazioni da eseguire in chiusura della applicazione. Supponiamo quindi di avere finito di scrivere il sorgente .java. Siamo sicuramente ansiosi di poterlo vedere in funzione perlomeno su di un emulatore! Avviamo il programma *kToolbar* del J2ME Wireless Toolkit, scegliamo di creare un nuovo progetto e chiamiamolo *SmsSender*. Indichiamo il nome della classe che estende MIDlet, nel nostro caso sempre *SmsSender*. Il toolkit ci chiederà di configurare il progetto, modificando qualche impostazione: innanzitutto la *"Target Platform"*, portandola a *"MIDP 1.0"*, poi, sotto la tab *"Optional"* i vari campi, in particolare la *"MIDlet-Icon"* che sarà l'icona che distinguerà la nostra applicazione dalle altre. Bisogna poi copiare, come ci viene indicato, i sorgenti nella cartella *apps\SmsSender\src* che è stata appena creata nella directory dove abbiamo installato il toolkit (solitamente *c:\WTK22*). Dovremo allo stesso modo mettere i file "risorsa" nella cartella *res*, nel nostro caso due immagini png: *logoBig.png* che utilizziamo nella midlet e *logo.png* che usiamo come icona dell'applicazione e le eventuali librerie (noi non ne abbiamo) nella cartella *lib*. Ora possiamo eseguire la compilazione del progetto cliccando su *"Build"*. Se tutto è andato a buon fine, basterà scegliere *"Run"* per vedere la nostra creatura fare i primi passi sull'emulatore. Una volta sicuri che la nostra applicazione non dia problemi al terminale, saremo sicuramente ansiosi di verderla in azione direttamente sul nostro cellulare. L'operazione di installazione è abbastanza semplice, per prima cosa bisogna far generare l'archivio (file *jar*) e il descrittore (file *jad*), cliccando su *"Package"* -> *"Create Package"* nel menù *Project*. Controllate nella cartella *bin* del progetto, troverete *SmsSender.jad* e *SmsSender.jar*, i due file necessari per l'installazione dell'applicazione. Ora dovete inviare al cellulare la midlet, operazione diversa da modello a modello. Potete inviarla via infrarossi, bluetooth, cavetto o scaricarla via wap dopo averne fatto l'upload su un server. A voi la scelta.

CONCLUSIONI

I più attenti avranno notato il contatore di caratteri in basso nella schermata iniziale, molto utile per sapere quanti caratteri avete ancora a disposizione. Questa è solo una delle possibili funzioni aggiuntive che potete inventarvi. Altre potrebbero essere un archivio per i messaggi salvati, una rubrica interna, uno "spezzettatore" che divida i messaggi più lunghi di 120 caratteri e chissà quanti altri. Comunque il conta-caratteri (che trovate nella versione presente sul CD allegato alla rivista) è di rapida implementazione. Si tratta di una *StringItem* che viene aggiornata, ogni volta che il *TextField* del messaggio viene modificato, da un *ItemStateListener* che ne verifica la *size()*. Ricordatevi di impostare correttamente il vostro telefono con la configurazione gprs di tipo web, prima di utilizzare l'applicazione. Per farlo basta cercare nel sito del produttore del vostro cellulare o del vostro operatore di telefonia.
Buon lavoro!

Luca Mattei

Miglioriamo la longevità e la flessibilità delle nostre applicazioni

Un'applicazione aperta ai plug-in

Un buon software deve prevedere la possibilità di estendere le proprie funzionalità tramite componenti esterni: scopriamo come includere questa opportunità nelle applicazioni Visual Basic

Le applicazioni moderne tendono sempre più a offrire l'apertura verso plug-in realizzati da terze parti. Il perché di questa scelta è piuttosto evidente, scrivere software orientato ai plug-in offre numerosi vantaggi, fra i quali:

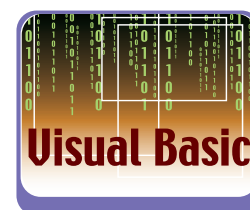
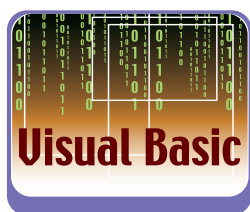
- **Un più semplice lavoro di squadra.** Mentre un team di programmatori lavora per rendere stabile l'applicazione di base, altre squadre possono curare lo sviluppo dei plug-in.
- **Maggiore longevità.** La possibilità di integrare componenti esterni è fondamentale in programmi che fanno uso di strumenti soggetti a rapido aggiornamento.
- **Migliore integrazione.** È possibile far condividere gli stessi plug-in ad applicazioni diverse, dando così l'impressione di una notevole coerenza interna ad una suite di prodotti intimamente differenti.
- **Specializzazione e personalizzazione.** Permettere a software house esterne di sviluppare aggiunte per il proprio software è la mossa migliore per consentire la distribuzione del proprio programma in paesi stranieri, per favorire l'integrazione di strumenti per accesso facilitato (come barre braille per i non-vedenti) o, più in generale, per garantire funzionalità specialistiche ad una determinata categoria d'utenti.

Anche una semplice applicazione gestionale può aumentare di molto la propria efficienza e versatilità avvalendosi di questo principio. Ma qual è la tecnica per implementare una funzionalità simile in Visual Basic? I passi da compiere possono essere sintetizzati come segue:

1. Individuare le necessità dell'applicazione di base in relazione ai componenti: ovvero stabilire che cosa è richiesto e cosa deve essere necessariamente fornito ai plug-in.
2. Scrivere un'interfaccia contenente metodi e proprietà prescelti, e salvarla in una libreria di classi ActiveX.
3. Scrivere il codice necessario per il caricamento e la gestione dei plug-in all'interno dell'applicazione di base.
4. Implementare ogni plug-in come una DLL ActiveX appartenente ad una classe (la stessa per tutti) che eredita dall'interfaccia prestabilita.

IL PROGETTO "PLUG 'N PAINT"

Come esempio pratico della tecnica sopra esposta, verrà creato un progetto molto semplice: il *"plug 'n paint"*. Realizzeremo un'applicazione di disegno molto simile allo storico Paintbrush di Windows, ma con in più la possibilità di estendere, mediante plug-in, gli strumenti a disposizione. Il software sarà dunque diviso in due: da una parte l'applicazione di base, che si preoccuperà di caricare dinamicamente i plug-in, verificarne il corretto funzionamento, e fungere da "direttore d'orchestra". Dall'altra i plug-in stessi (pennello, gomma, spray, etc…), che verranno richiamati opportunamente dall'applicazione di base, e disegneranno sulla tela. Per sviluppare l'applicazione, seguiremo fedelmente i quattro passi esposti nel paragrafo precedente. La nostra applicazione di base esporrà solo le funzionalità strettamente necessarie alla corretta gestione dei plug-in: lo sviluppo di soluzioni tipiche di questo genere di applicazioni (uso di buffer multipli, selezione di regioni, wrapper GDI, undo-redo, e via elencando) appesantirebbe solo gli esempi ed esulerebbe dallo scopo della trattazione. Tutto ciò di cui abbiamo bisogno è una ToolBar *(TBarTool)* per la selezione degli strumenti, un ImageList *(ImListTool)* per il caricamento

NOTA

Sul CD sono presenti due versioni del progetto di esempio. La prima è relativa all'implementazione iniziale di ITool. La seconda (in PnP2), inerente l'espansione del metodo di comunicazione, presenta anche la realizzazione di alcuni strumenti aggiuntivi, come lo spirografo.

delle rispettive immagini, un *FileListBox (FileTool)* per rintracciarne velocemente i file sul disco, e un PictureBox *(PBoxTela)* che costituirà la nostra area di disegno. In Fig. 2, è possibile ammirare il nostro spartano form di partenza.

NECESSITÀ DELL'APPLICAZIONE

Ora che sappiamo com'è fatta la nostra applicazione di base, si tratta di passare al primo punto della lista: individuare i compiti dei plug-in.
Questi dovranno fornire al programma principale: un nome che li identifichi (dal momento che tutti appartengono alla stessa classe); un'icona da associare al relativo pulsante sulla toolbar e un "property form" relativo alle opzioni dello strumento, simile a quello che si ottiene premendo il tasto *F4* di Visual Basic. L'applicazione di base, invece, dovrà preoccuparsi di notificare al plug-in correntemente selezionato i principali eventi che interessano la tela: il *MouseMove*, il *MouseDown* e il *MouseUp* sono sufficienti.

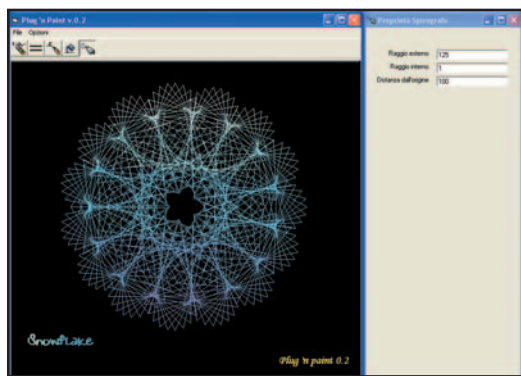



Fig. 1: L'applicazione d'esempio Plug 'n Paint, si basa interamente su plug-in per disegnare forme diverse, come questo fiocco di neve

CREAZIONE DELL'INTERFACCIA

Tutto quello che abbiamo stabilito nel paragrafo precedente è il minimo indispensabile perché un plug-in riesca a svolgere decentemente il proprio lavoro. Una volta stabilita la struttura, questi eventi devono necessariamente essere definiti da ciascun metodo, dal momento che la mancanza di uno soltanto di questi elementi potrebbe mandare in crash l'applicazione: cosa succederebbe, ad esempio, se venisse chiamato un metodo *Tool.Mousemove()*, quando il plug-in non lo espone? Dobbiamo, quindi, fornire il prototipo delle funzioni e delle proprietà che uno strumento dovrà necessariamente implementare. Questo sembra un modello ragionevole:

Fig. 2: I componenti essenziali per la prima realizzazione del form

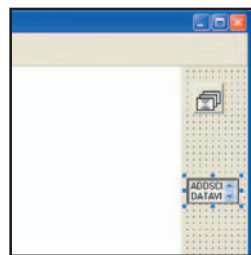
```
Public PBoxTela As PictureBox
Public Nome As String
Public PropertyForm As Form
Sub MouseDown(Button As Integer, Shift As Integer, X
        As Single, Y As Single)
End Sub
Sub MouseUp(Button As Integer, Shift As Integer, X
        As Single, Y As Single)
End Sub
Sub MouseMove(Button As Integer, Shift As Integer, X
        As Single, Y As Single)
End Sub
Function GetIcon() As IPictureDisp
End Function
```

Quella che abbiamo appena scritto prende il nome di '*interfaccia*'. Un'interfaccia è, uno scheletro che si limita a stabilire i metodi e le proprietà che una classe derivata deve necessariamente esporre, ma lascia a quest'ultima l'onere della loro implementazione. Abbiamo, insomma, stabilito un 'contratto' fra la nostra applicazione e il plug-in, che quest'ultimo è tenuto a rispettare. Per completare il secondo punto, però, dobbiamo ancora salvarla in una libreria di classi.

CREAZIONE DELLA LIBRERIA

Dal momento che l'interfaccia stabilita nel paragrafo precedente dovrà essere utilizzata nella nostra applicazione e in tutti i plug-in, è fondamentale salvarla in una libreria di classi *(DLL ActiveX)*, che ne favorisca il riutilizzo. Bisognerà dare a quest'interfaccia un nome appropriato: *ITool* è una buona idea (la prassi comune è che il nome delle interfacce inizi con la lettera *'I'*). Apriamo, quindi, una nuova istanza di Visual Basic, scegliamo il progetto *ActiveX DLL*, modifichiamo il nome della classe *"Class1"*, in *"ITool"*, e all'interno di questo modulo scriviamo la struttura riportata nel paragrafo precedente. Dobbiamo dare un nome facilmente riconoscibile a questa libreria, quindi andiamo in *Project/Properties*, e nel campo *"descrizione"* scriviamo *"Plug 'n Paint Type Library"*. Una volta compilata *(File /Make)*, questa libreria si rivelerà una struttura portante: dovremo ricordarci di includerla nell'SDK che forniremo agli sviluppatori esterni! (Files su CD: *PPInterfaces.Vbp, ITool.Cls, PPInterfaces.DLL,* da registrare mediante *RegSvr32 .exe).*

CARICARE I PLUG-IN

Al terzo punto della scaletta proposta all'inizio, troviamo l'implementazione di una routine fondamentale, appartenente all'applicazione di base: quella relativa al caricamento degli strumenti. Prima di scriverla dobbiamo assicurarci di includere nel progetto il riferimento alla libreria che abbiamo costruito nel paragrafo precedente, andando su *Project/ References* e spuntando la libreria in questione. Ora possiamo scrivere la routine che, per semplicità, qui è associata direttamente all'evento *Form_Load.*

```
Private Sub Form_Load()
  FileTools.Path = App.Path & "\Strumenti"
  'Carichiamo i componenti
  Dim i As Integer
  For i = 0 To FileTools.ListCount - 1
    'Registriamo il componente in modo invisibile
    Shell "regsvr32 -s """ & FileTools.Path & "\" &
                              FileTools.List(i) & """"
    'Prendiamo il nome del componente (senza l'estensione)
    Dim NomeFile As String
    NomeFile = Left(FileTools.List(i), Len(FileTools.List(i)) - 4)
    'Carichiamo l'oggetto in una variabile d'appoggio
    Dim tempTool As ITool
    Set tempTool = CreateObject(NomeFile & ".CTool")
    'Aggiungiamo l'oggetto alla collection
    Tools.Add tempTool
    Set Tools(i + 1).PBoxTela = PBoxTela
    [...] segue codice relativo al caricamento delle immagini
  Next
End Sub
```

Qui sopra è riportato un lungo estratto (è stata eliminata la gestione delle icone sulla toolbar) della routine di caricamento, il cui codice integrale è presente nel file allegato su CD. Vale la pena di leggere con attenzione questo codice, visto che è il cuore del sistema. Ogni plug-in andrà salvato nella sottocartella "*strumenti*", ed è a quella che punta *FileTools*. Mediante un ciclo, ogni DLL viene registrata con *regsvr32*: un prassi comune che garantisce il funzionamento di un componente anche se questo non è stato configurato tramite installer. È anche consigliato deregistrare i componenti con un altro ciclo associato all'evento *Form_QueryUnloade*, mediante una chiamata a *RegSvr32 –u*, in modo da non appesantire inutilmente il sempre stracarico registro di configurazione di Windows. Il componente viene quindi caricato con *TempTool = CreateObject(nome .CTool)*, il che impone:

1. Che il plug-in sia un ActiveX DLL.
2. Che la classe di ogni plug-in dovrà chiamarsi *CTool*.
3. Che il nome del file ogni plug-in dovrà coincidere col di progetto dello stesso.
4. Che la classe *CTool* implementi l'interfaccia *ITool*.

Dovremo tener conto di tutti questi vincoli quando scriveremo i componenti, e farli presenti agli sviluppatori esterni. È interessante notare che si è usata una variabile temporanea di tipo *ITool (TempTool)* per il valore di ritorno di *CreateObject*, cosa che impone all'istruzione un *Query Interface* per stabilire se la conversione sia possibile (da cui la necessità del vincolo **4**). Nel momento in cui questa avviene con successo, disponiamo di un oggetto in cui è possibile usare i metodi della proprietà *ITool*, ma con l'implementazione particolare stabilita in *CTool*. Potenza del polimorfismo! Dopo questa routine, il lavoro è tutto in discesa, dal momento che disponiamo di una collection di oggetti che espongono la stessa interfaccia. Tutto quello che dobbiamo fare è passare gli eventi di *PboxTela* allo strumento correntemente selezionato (in indice).

```
Private Sub PBoxTela_MouseMove(Button As Integer,
                 Shift As Integer, X As Single, Y As Single)
   Tools(indice).MouseMove Button, Shift, X, Y
End Sub
```

Qui sopra è riportato questo passaggio relativamente al metodo *MouseMove;* discorso identico (e stesso codice) vale per i metodi *MouseUp* e *MouseDown*. *[File su CD: BaseApp.vbp, BaseApp.frm]*

CREAZIONE DEL PLUG-IN

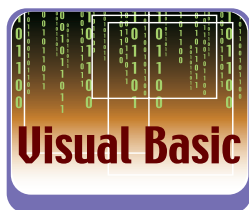
Il codice scritto nella sezione precedente impone gran parte delle scelte nella costruzione del plug-in. Questo non è necessariamente un male, anzi! Quando si fornisce un modello per sviluppatori esterni, non c'è niente di meglio che la standardizzazione di un metodo fisso e ben definito. Nel creare il nostro primo plug-in (lo strumento *"Matita"*) vedremo che saranno proprio i vincoli esposti nel paragrafo precedente a farci da guida. Per il punto **1** apriamo una nuova istanza di Visual Basic e selezioniamo ActiveX DLL come progetto. Per il punto **2** rinominiamo la classe predefinita *Class1* in *CTool* Per il punto **3** andiamo in *Project/Properties* e cambiamo il nome del progetto in *"Matita"*. Aggiungiamo al progetto un form SDI *(PropForm)* che fungerà da form delle proprietà. Ora dobbiamo soddisfare il punto **4**, ovvero far implementare a *CTool* l'interfaccia *ITool*. Per questo, dopo aver stabilito una referenze di progetto ad *ITool* (come nell'applicazione di base) basta inserire all'interno della classe la riga

```
Implements ITool
```

Questo è sufficiente per creare dal nulla l'oggetto *ITool*, del quale – per "contratto" – siamo obbligati a implementare tutti i metodi.

```
Implements ITool
Private oldx As Long, oldy As Long
Private m_Tela As PictureBox
Private m_Nome As String
Private m_PropertyForm As Form
Public Sub Class_Initialize()
   m_Nome = "Matita"
   Set m_PropertyForm = PropForm
   oldx = -1: oldy = -1
```

NOTA

Spesso i programmi più avanzati non si limitano a fornire una base di tipi, ma un intero sistema di sviluppo, o un API, che permetta un'integrazione proprietaria: sovente, per scaricare l'SDK, viene richiesta l'adesione (a pagamento) a un network di sviluppatori. Questo permette la nascita di società che si occupano esclusivamente di progettazione di plug-ins per un programma specifico.

```
End Sub
Public Function ITool_GetIcon() As stdole.Picture
    Set ITool_GetIcon = m_PropertyForm.Icon
End Function
Public Sub ITool_MouseDown(Button As Integer, Shift
                        As Integer, X As Single, Y As Single)
    oldx = X: oldy = Y
End Sub
Public Sub ITool_MouseUp(Button As Integer, Shift As
                        Integer, X As Single, Y As Single)
    oldx = -1: oldy = -1
End Sub
Public Sub ITool_MouseMove(Button As Integer, Shift
                        As Integer, X As Single, Y As Single)
    If oldx = -1 And oldy = -1 Then Exit Sub
    If Button = vbLeftButton Then m_Tela.Line (oldx,
                                        oldy)-(X, Y)
    oldx = X: oldy = Y
End Sub
```

Il codice riportato qui sopra è la più semplice implementazione possibile del nostro strumento. Ogni commento è superfluo, ad eccezione delle seguenti considerazioni:

- Come avviene per ogni oggetto, i metodi di *ITool* vengono dichiarati anteponendo il nome dell'oggetto seguito da underscore (ovvero: *Sub ITool_Metodo*)
- Per esportare l'icona dello strumento abbiamo impostato un riferimento a quella del form delle proprietà (che assoceremo, quindi, all'immagine di una matita)
- Non abbiamo usato i campi definiti nell'interfaccia, bensì delle variabili *private*.

L'ultimo punto è degno di nota. Quando si implementa un'interfaccia, infatti, i campi da questa esposti divengono automaticamente delle proprietà, con relativi metodi *(Property Get/Set)*. È quindi necessario creare delle variabili private a cui associare le relative proprietà. Questo, ad esempio, è il codice relativo alla proprietà *Nome:*

```
Public Property Let ITool_Nome(ByVal RHS As String)
    m_Nome = RHS
End Property
Public Property Get ITool_Nome() As String
    ITool_Nome = m_Nome
End Property
```

Dopo aver ripetuto questo noioso lavoro anche per le altre due proprietà (*PBoxTela* e *PropertyForm*), possiamo compilare *(File/make)*, il nostro strumento che siamo tenuti a chiamare *"Matita.DLL"*, e a salvare nella sottodirectory *Strumenti*. (File su CD: *ToolMatita.vbp*, *ToolMatita.cls*, *ToolMatita.frm*, *Strumenti/Matita.DLL).*

Nella scelta dell'architettura delle plug-in, occorre considerare anche il rischio a cui si espone il sistema nel caso di un componente 'malevolo': un'interfaccia è sufficiente a obbligare l'uso di una struttura minima esterna, ma all'interno del codice associato è possibile prendere il totale controllo della macchina. Questo ci insegna a scaricare plug-in solo da fonti fidate, riconosciute, e con firma digitale - se non a codice aperto.

COMUNICAZIONE BILATERALE

Dopo tanto lavoro, possiamo finalmente vedere la matita tracciare i suoi tratti sulla tela. Se ora volessimo sbizzarrirci, potremmo inventare migliaia di strumenti per la nostra applicazione: pennelli, carboncini, aerografi con antialiasing, forme evolute, e via discorrendo… Tutto questo è molto divertente, ma in questa sede dobbiamo occuparci di cose più serie, come, ad esempio, rivedere il nostro modello d'interfaccia (dopo tanto lavoro!). Sussiste una serie di motivi per i quali il nostro sistema non è completo: innanzitutto la comunicazione applicazione-> plug-in è essenzialmente unilaterale: quando l'applicazione chiama il plug-in, questo può solamente scrivere sulla tela, senza avere nessuna informazione dal chiamante. E se, ad esempio, volessimo aggiungere i colori al nostro disegno, per ora forzatamente monocromatico? Dovremmo passarli nel messaggio o complicare ulteriormente l'interfaccia *ITool*?

Tutti questi sistemi sono possibili, ma sconsigliati: quando non complicano il codice, infatti, rendono impossibile il riutilizzo. L'idea da seguire, invece, è di creare un oggetto *appObject* che rappresenti, agli occhi dei plug-in, i servizi offerti dall'applicazione di base. Per fare un esempio pratico, creeremo il nostro secondo strumento: *la linea*

I PROBLEMI DEL SINGLE-BUFFERING

Perché mai una linea dovrebbe causare tanti e tali problemi da imporre delle modifiche al modello d'interfaccia? In effetti, le differenze fra uno strumento *"linea"* e una "matita" sono davvero poche! In teoria basterebbe togliere l'aggiornamento dei valori *oldx* e *oldy*, nell'evento *MouseMove*, facendolo diventare:

```
Public Sub ITool_MouseMove(Button As Integer, Shift
                        As Integer, X As Single, Y As Single)
    If oldx = -1 And oldy = -1 Then Exit Sub
    If Button = vbLeftButton Then m_Tela.Line (
                                    oldx, oldy)-(X, Y)
End Sub
```

In realtà il risultato di quest'intervento maldestro è ben differente, e visibile in Fig. 3: abbiamo inventato un nuovo strumento molto singolare, ma non è certo una linea! Il problema si verifica perché ad ogni movimento del mouse, non corrisponde un ridisegno della tela, ovvero un suo ripristino allo stato antecedente la chiamata al plug-in. Sarebbe di certo possibile inserire un PictureBox all'interno del componente, in cui salvare lo stato del disegno al

primo evento *MouseDown*, ma significherebbe sprecare inutilmente memoria e codice. In realtà, questo servizio è molto più efficace se offerto dall'applicazione di base, la quale si suppone, peraltro, abbia già un bel po' di copie di scorta per le operazioni di undo. Per questo modificheremo leggermente l'applicazione di base, inserendo nel form un buffer secondario, (nella fattispecie, un secondo PictureBox di nome *PBoxCopia*, invisibile, delle stesse dimensioni di PBoxTela). Ora disponiamo di un backbuffer di appoggio: grazie all'API *BitBlt* potremo passarne il contenuto al buffer primario.

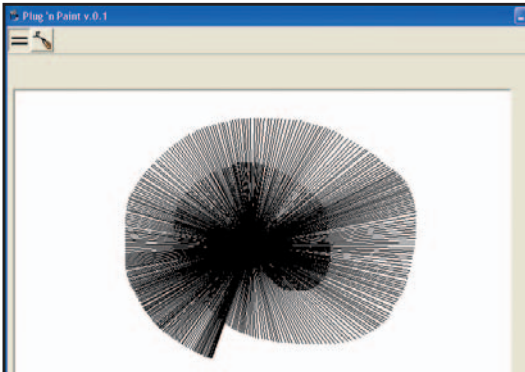



Fig. 3: ***Il pasticcio di linee derivante dal mancato ridisegno della schermata ad ogni MouseMove. Problemi come questo si risolvono con un corretto backbuffering***

L'OGGETTO APPOBJECT

Ora che disponiamo di due buffer possiamo pensare al nuovo modello di comunicazione fra plug-in e applicazione di base. Sembra ragionevole pensare di passare ancora *PBoxTela* al plug-in, ma concedendo a quest'ultimo la possibilità di richiedere un ridisegno del buffer (ovvero un trasporto di *PBoxCopia* in *PBoxTela*). Quando il plug-in avrà finito, e sarà sicuro del risultato, potrà notificare all'applicazione il compimento del lavoro. Possiamo, quindi, stabilire quest'interfaccia:

```
Sub RequestRedraw()
End Sub
Sub NotifyJobDone()
End Sub
```

Sono poche righe, ma ci saranno molto utili: andranno inserite in un nuovo modulo di classe (*IApplication*) all'interno della nostra *Type Library.*
Già che ci siamo, diamo anche un piccolo ritocco all'interfaccia *ITool*, aggiungendo l'oggetto:

```
Public AppObject As IApplication
```

Questo sarà a disposizione di ogni strumento, e permetterà i servizi di richiesta di ridisegno e la notifica del lavoro compiuto. Una volta compilata la libreria, possiamo passare all'applicazione di base implementando l'interfaccia, mediante l'aggiunta al codice della riga:

```
Implements IApplication
```

Questo ci permetterà di definire i metodi antitetici (basati sulla *GDI-API BitBlt*):

```
Private Sub IApplication_NotifyJobDone()
    With PBoxCopia
        BitBlt .hDC, 0, 0, .ScaleWidth,
            .ScaleHeight, PBoxTela.hDC, 0, 0, vbSrcCopy
    End With
End Sub
Private Sub IApplication_RequestRedraw()
    With PBoxTela
        BitBlt .hDC, 0, 0, .ScaleWidth, .ScaleHeight,
                    PBoxCopia.hDC, 0, 0, vbSrcCopy
    End With
End Sub
```

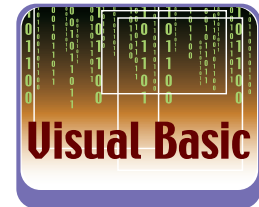
Il primo fissa sul backbuffer il lavoro eseguito sul buffer primario, il secondo fa esattamente l'opposto. Inoltre, dovremo far puntare la variabile oggetto *appObject* di ogni plug-in all'applicazione stessa, all'interno del ciclo di caricamento in *Form_Load*, così:

```
Set Tools(i + 1).appObject = Me
```

Dovremo infine occuparci di ridefinire sia la matita che la linea, aggiungendo la variabile private *m_appObject* e le rispettive *[property Get/Set]*, come abbiamo imparato qualche paragrafo fa.

```
Public Sub ITool_MouseUp(Button As Integer, Shift As
                    Integer, X As Single, Y As Single)
    oldx = -1: oldy = -1
    m_AppObject.NotifyJobDone
End Sub
Public Sub ITool_MouseMove(Button As Integer, Shift
                    As Integer, X As Single, Y As Single)
    If oldx = -1 And oldy = -1 Then Exit Sub
    m_AppObject.RequestRedraw
    m_Tela.Line (oldx, oldy)-(X, Y)
End Sub
```

Ce l'abbiamo fatta: la nuova linea è esattamente come la precedente ma ha in più una richiesta di ridisegno schermo per ogni *MouseMove*, ed una notifica di operazione compiuta per ogni *MouseUp*.
Sono sicuro che tutte queste noiosissime modifiche sono servite da lezione per chiarire un concetto fondamentale: occorre studiare molto un'interfaccia prima di implementarla!

Roberto Allegra
roberto.allegra@ioprogrammo.it

NOTA

Quando si crea una interfaccia, questa dovrebbe essere ben progettata, in maniera da 'resistere' elasticamente alle future esigenze dell'applicazione. Qualora un cambiamento sia inevitabile, è sempre buona norma lasciare immutata la classe o il metodo che si è deciso di modificare, e crearne uno nuovo, terminante per *Ex* (Es: *IToolEx*). Questo non risolve del tutto i problemi del "Dll hell", ma tenta di porre un freno all'incompatibilità fra versioni diverse.

Convertire le GUI create in VB in codice Java

Interfaccie grafiche da VB a Java

Grazie al programma che svilupperemo sarà possibile trasformare il codice dei Form di Visual Basic in codice Java per realizzare interfaccie grafiche Swing

Il progetto che ci proponiamo di realizzare con il presente articolo è quello di realizzare un programma interamente scritto in Java che possa leggere i file sorgenti con estensione *.frm* prodotti dall'ambiente Visual Basic per poter codificare le interfaccie create in tale ambiente e trasformare il tutto in codice Java. Questo risulterà sicuramente utile a quanti di voi, avendo realizzato programmi in VB, volessero trasportare tutto nel linguaggio di Sun. Chi non avesse dimestichezza nella programmazione di GUI, troverà sicuramente utile la lettura del codice in cui sono presenti numerosi spunti e spiegazioni dei metodi più comunemente usati per realizzare applicazioni grafiche.

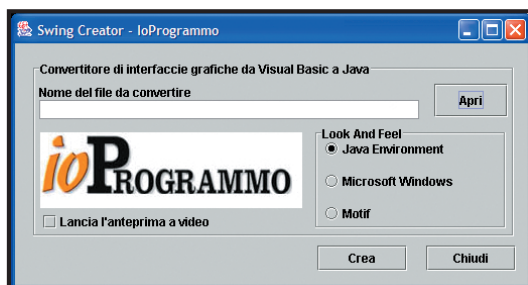



Fig. 1: ***Il programma in funzione***

UN SGUARDO AI FORM VB

Prima di cominciare è bene dare uno sguardo a come Visual Basic costruisce il codice per realizzare la grafica a video. Chi già programma in VB, solitamente realizza le proprie interfacce senza preoccuparsi di cosa ci sia dietro. Evitando di entrare in dettagli troppo tecnici, diamo comunque un'occhiata a come viene codificato il codice di ciò che vediamo a video. Questo è indispensabile, in quanto il nostro programma non farà altro che leggere sequenzialmente queste istruzioni e tramutarle in istruzioni valide per la JVM.

Ogniqualvolta si avvia l'ambiente di sviluppo, viene creato un nuovo form di default. Questo rappresenta la finestra in cui lo sviluppatore andrà a inserire gli oggetti presenti sulla barra di sinistra. Il codice che VB genera a seguito delle operazioni dello sviluppatore non è visibile tramite la finestra *Codice*, ma se noi volessimo sapere come la nostra GUI è stata codificata dovremmo salvare il form in questione (ricordo che i form hanno tutti estensione *.frm*) e aprirlo tramite un semplice editor di testo (Blocco note di Windows va benissimo per questo scopo!). Ci accorgeremmo che ogni oggetto viene codificato in VB tramite blocchi di codice contenuti nelle parole chiave *"Begin VB."* e *"End"*. All'interno, inserite su più righe, sono presenti le varie proprietà per l'oggetto che si è disegnato. Ad esempio, se si volesse cambiare il testo di un pulsante, lo si potrebbe fare all'interno di questo file editando la riga in cui compare la scritta *"Caption"* ed inserendo il testo dopo il simbolo =.

Il nostro programma, quindi, non dovrà fare altro che aprire il file scelto, leggere ogni singola riga di te-

SwingCreator - Blocco note

File Modifica Formato Visualizza ?

```
VERSION 5.00
Begin VB.Form SwingCreator
   Caption         =   "Swing Creator - IoProgrammo"
   ClientHeight    =   3615
   ClientLeft      =   60
   ClientTop       =   450
   ClientWidth     =   7665
   LinkTopic       =   "Form1"
   ScaleHeight     =   3615
   ScaleWidth      =   7665
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton cmdCrea
      Caption         =   "Crea"
      Height          =   375
      Left            =   6120
      TabIndex        =   1
      Top             =   3000
      Width           =   1335
   End
   Begin VB.Frame panel1
      Caption         =   "Convertitore di interfaccie grafiche da
      Height          =   2655
      Left            =   240
      TabIndex        =   0
      Top             =   240
      Width           =   7215
      Begin VB.CheckBox chkAnteprima
         Caption         =   "Lancia l'anteprima"
         Height          =   255
         Left            =   120
         TabIndex        =   9
         Top             =   2280
         Width           =   3855
      End
```

Fig. 2: ***Ecco come appare un file .frm una volta aperto con un editor di testo qualsiasi***

sto. Ogniqualvolta ci si trova di fronte ad una riga che inizia per *"Begin VB."* sapremo che quello è un nuovo oggetto da disegnare nella finestra. A questo punto, dovremo leggere ogni riga per recuperare le proprietà dell'oggetto finché non si arriverà alla riga riportante la parola *"End"*. Le cose tendono a complicarsi leggermente quando, all'interno di un oggetto, troviamo un altro oggetto. Questo può essere il caso in cui un oggetto creato (quale ad esempio un *TextBox*) sia figlio di un altro oggetto (come un Frame). Non preoccupiamoci troppo! Risolveremo il problema tramite una funzione ricorsiva.

GLI OGGETTI TRATTATI

Nella prima parte di questo articolo realizzeremo lo scheletro del nostro programma. Avremo così la possibilità di leggere GUI che contengono gli oggetti di Tab. 1.

Visual Basic	Java
CommandButton	JButton
Label	JLabel
CheckBox	JCheckBox
OptionButton	JRadioButton
Frame	JPanel
TextBox	JTextField

Tabella 1: Le più semplici associazioni fra i due linguaggi

Sono stati schematizzati i vari oggetti con il nome della classe in VB ed il nome della classe in Java. Il prossimo mese completeremo il tutto inserendo la codifica dei controlli *"Image"* e *"Picture"*, dei menù e tratteremo il raggruppamento degli *OptionButton* che, come molti di voi sapranno, in Java devono essere associati ad un oggetto della classe *ButtonGroup* per poter essere legati tra di loro.

TRE CLASSI FONDAMENTALI

Per il nostro progetto ci avvaleremo di tre classi fondamentali: *Oggetto, ScriviFile, SwingCreator*.
Ogni qualvolta incontreremo un nuovo oggetto nel file *.frm,* creeremo un'istanza di questa classe. Tale oggetto conterrà tutte le proprietà necessarie alla sua corretta visualizzazione. Quando vorremo passare alla creazione del file *.java,* avremo un array contenente tutti gli oggetti da scrivere.
La classe è composta da un insieme di proprietà tutte dichiarate di tipo *String* con il qualificatore *private*. È stato scelto di utilizzare tutte stringhe in quanto il valore di queste proprietà dovrà essere trascritto su file, quindi era inutile ogni volta effettuare un cast per tutti quei valori che dovrebbero essere numerici. Il qualificatore *private*, invece, ci permette di rendere inaccessibili tali dati dall'esterno della classe se non utilizzando i metodi *get* per leggere il valore, e *set* per impostarne uno nuovo. Il metodo *get* ritorna una stringa mentre il metodo *set* accetta una stringa come unico parametro. Da notare inoltre, alcuni metodi *set* un po' particolari. Ad esempio:

```
public void setWidth(String width) {
  int temp = Integer.parseInt(width) / 15;
  this.width = "" + temp;
}
```

Prima di registrare il valore nella variabile corrispondente, tale valore viene diviso per 15. Questo perché VB di default non utilizza i pixel come unità di misura bensì i *twips* che sono appunto 1/15 dei pixel.
Un'altra particolarità ci viene data dai due metodi *setClientWidth* e *setClientHeight:*

```
public void setClientWidth(String clientWidth) {
  int temp = (Integer.parseInt(clientWidth) / 15) + 6;
  this.clientWidth = "" + temp; }
public void setClientHeight(String clientHeight) {
  int temp = (Integer.parseInt(clientHeight) / 15) + 32;
  this.clientHeight = "" + temp;
}
```

Nel metodo *setClientWidth*, al valore finale viene aggiunto 6, così come nel metodo *setClientHeight* viene aggiunto 32. Questo perché, mentre in Java la dimensione della finestra è data da tutto ciò che viene disegnato a video, in VB questa dimensione rappresenta l'area cosiddetta *Client,* cioè il pannello dove è possibile disegnare gli oggetti, ESCLUSO la zona dei Menu. Gli oggetti istanziati dalla classe *Oggetto* posseggono un unico costruttore:

```
public Oggetto(String tipo, String nome) {
  this.tipo = tipo;
  this.nome = nome;
  alignment = "";
  caption = "";
  enabled = "";
  text = "";
  toolTipText = "";
  value = "";
  visible = "";
}
```

Tale costruttore accetta due argomenti che sono rispettivamente il tipo di oggetto che andremo a creare ed il nome con il quale verrà identificato all'interno del codice. Queste due variabili andranno ad essere inserite all'interno delle rispettive proprietà; sempre nel costruttore, inoltre, ci preoccuperemo di istanziare le proprietà che potrebbero non avere un valore all'interno del file *.frm.* La seconda classe che andiamo ad analizzare è la classe *ScriviFile.* Attraver-

WRAPPERS

Le classi *BufferedReader* e *BufferedWriter* utilizzano un buffer per memorizzare le informazioni da leggere e scrivere. Questo permette di ridurre gli accessi fisici al file. Tali classi vengono chiamate wrappers in quanto incapsulano all'interno oggetti delle classi *FileReader* e *FileWriter* estendendone le funzionalità.

ISTANZA

Con il termine *istanza* si intende la creazione fisica dell'oggetto. In pratica si ha un'istanza ogniqualvolta, dopo essere stato dichiarato, all'oggetto viene chiamata l'istruzione *new*.

so un'istanza di questa classe, si avrà la possibilità di creare il file *.java* contenente il codice della GUI estrapolata dal file di VB.
I metodi di questa classe sono in tutto sei:

```
1. public void intestazione(Oggetto obj[]);
2. public int dichiarazioni(Oggetto obj[]);
3. public void inserisciOggetto(Oggetto obj);
4. public void pieDiPagina(Oggetto obj[]);
5. public void setLookAndFeel(int laf);
6. public void close();
```

Vediamo inanzitutto come creare un file in Java dopodichè analizzeremo i sei metodi esposti.

I FILE IN JAVA

Per lavorare con i file in Java bisogna creare un canale attraverso il quale poter leggere oppure scrivere. Esistono svariate classi che ci permettono di fare questo. Nell'esempio seguente abbiamo creato il file utilizzando la seguente sintassi:

```
file = new File(percorso);
pw = new PrintWriter(new BufferedWriter(
                              new FileWriter(file)));
```

L'oggetto *file* fa riferimento ad un file fisico. Per istanziarlo al costruttore va passato il percorso del file che si vuole leggere oppure che si vuole creare.
Dopodichè abbiamo istanziato l'oggetto *pw* della classe *PrintWriter*. Tale classe ci permette di scrivere su file utilizzando gli stessi metodi della classe *System.out*. Quindi, se si volesse scrivere una stringa, la sintassi del comando sarebbe: *pw.println(stringa);*.
Al costruttore abbiamo passato un oggetto della classe *BufferedWriter* che incapsula un oggetto della classe *FileWriter* che, a sua volta, incapsula l'oggetto file creato in precedenza. Sicuramente vi starete domandando come mai viene utilizzata questa tecnica di scatole cinesi. La risposta è semplice: per istanziare un oggetto della classe *FileWriter* abbiamo bisogno di un oggetto *File*. Tale classe, però, ci permette solamente di scrivere un carattere alla volta.
Possiamo ovviare a questo, istanziando un oggetto della classe *BufferedWriter* che possiede un metodo chiamato *writeLine()* per poter scrivere un'intera sequenza di caratteri. Per scrivere testo formattato, però, abbiamo bisogno di un oggetto *PrintWriter* che, come visto sopra, accetta come argomento al costruttore un *BufferedWriter.* Tutto chiaro allora?
Come vedremo in seguito, per leggere dal file *.frm* ci fermeremo alla classe *BufferedReader* che con il suo metodo *readLine()* ci permette di leggere un'intera riga.

Fig. 3: ***Tramite il programma è possibile codificare anche interfaccie complesse che altrimenti avrebbero richiesto molto tempo nello sviluppo***

CREIAMO IL FILE JAVA

Vediamo, ora, come poter utilizzare i metodi di questa classe per creare un file java contenente il codice della GUI. Come prima cosa, per creare il file, va istanziato un oggetto di questa classe dopodichè vanno chiamati i vari metodi nell'ordine esposto sopra. I metodi *dichiarazioni, intestazione, pieDiPagina* richiedono come parametro un array di oggetti *Oggetto*. Tale vettore conterrà tutti gli oggetti creati che andranno inseriti nella nostra interfaccia.
Il metodo *intestazione* fa da cappello al file java in quanto inserisce i vari package oltre a chiamare la classe con il primo oggetto dell'array (che sarà l'oggetto Form di VB). Il metodo *dichiarazioni* permette, riconoscendo i singoli oggetti, di inserirne la dichiarazione nel file. In questa fase avviene la prima conversione: ogni oggetto dell'array viene confrontato all'interno di un ciclo *if*, con una stringa contenente i tipi riconosciuti dal programma. Se viene trovata una corrispondenza, viene inserito nel file il codice appropriato. Ad esempio, se l'oggetto preso in questione fosse un TextBox di nome *txtPercorso* avremo che quando la JVM incontra l'istruzione

```
else if(obj[i].getTipo().equals("TextBox"))
   pw.println(" private JTextField " + obj[i].getNome()
                                              + ";");
verrà inserito nel file la stringa: private JtextField
                                          txtPercorso;
```

Il metodo conclude inserendo un riferimento al *Container*, la classe che in *Swing* fa riferimento al pannello su cui verranno disegnati gli oggetti. Questo è da tenere a mente, infatti, quando analizzeremo l'inserimento degli oggetti, ogni volta che un oggetto avrà come padre il *Form*, andrà aggiunto al *Container* con l'istruzione *c.add(oggetto)*.
Per aggiungere gli oggetti al file viene chiamato il metodo *inserisciOggetto* al quale passiamo come parametro l'oggetto da inserire (e non l'array completo). Ogni oggetto viene fatto ciclare all'interno di

SWING
È la classe di Java per la creazione di interfaccie grafiche. Estende le funzionalità della vecchia *awt* la quale ha come difetto il fatto di essere dipendente dal sistema su cui gira l'applicazione. Quasi tutte le classi di oggetti grafici quali pulsanti, etichette,etc. sono state riscritte in *swing* interamente in Java. Trovate i sorgenti in *\src\javax\swing*

GUI (GRAPHICAL USER INTERFACE)
Con questo termine si vuole indicare tutte le finestre disegnate a video che permettono un'interazione con l'utente che le usa. Ogni programma è dotato di finestre attraverso le quali l'utente effettua le sue scelte. Windows stesso permette di accedere alle informazioni del proprio computer tramite GUI (Il desktop, ad esempio).

istruzioni *if/else* che controllano la proprietà tipo. Se l'oggetto viene riconosciuto correttamente, vengono inserite nel file le istruzioni in Java adeguate. In questa prima versione del programma, le proprietà riconosciute sono:

Alignment
Caption
Enabled
Text
ToolTipText
Visible

Chiude il metodo il seguente codice:

```
// Aggiunta dell'oggetto al pannello
if(obj.getTipoPadre().equals("Form"))
   pw.println(spazio + "c.add(" + obj.getNome() + ");");
else
   pw.println(spazio + obj.getPadre() + ".add(
                                  " + obj.getNome() + ");");
```

Per comprendere tale codice è indispensabile sprecare due righe nella spiegazione di come sono costruite le interfaccie in Java. Per essere brevi si potrebbe paragonare il Form di VB al Container di Java. Tale Container definisce l'area della finestra in cui è possibile inserire i vari oggetti grafici quali pulsanti, etichette, etc. Essendo *Container* la superclasse di tutti gli oggetti grafici possiamo dire che anche questi ultimi sono essi stessi contenitori, quindi, volendo, è possibile inserire, ad esempio, un pulsante all'interno di un pulsante. Questo fa sì che si crei una gerarchia Padre/Figlio dove il padre è appunto il contenitore che ospita l'oggetto *Figlio.* Tale gerarchia è evidente tra il Container ed i vari oggetti; infatti per disegnare un oggetto a video si utilizza il metodo *add()* applicato al container; ma è anche possibile trovarsi di fronte ai casi in cui l'oggetto venga disegnato all'interno di un Frame (*JPanel* in Java). In questo caso va prima aggiunto l'oggetto al pannello con

```
nomePannello.add(oggetto);
```

dopodichè verrà aggiunto il pannello al container con

```
nomeContainer.add(pannello);
```

Nel codice sopra riportato è evidente che nel caso in cui la proprietà *tipoPadre* è uguale a *"Form",* tale oggetto verrà aggiunto direttamente al *Container;* in caso contrario verrà aggiunto all'oggetto *padre.*
Gli ultimi due metodi di questa classe sono *setLookAndFeel* e *close.* Il primo sfrutta una caratteristica delle interfaccie *Swing,* quella, cioè, di poter disegnare le GUI a video mantenendo l'aspetto tipico di Java (settato di default), oppure assumendo l'aspetto tipico delle interfaccie Windows o Linux. Il secondo, invece, chiude il file e va chiamato alla fine del programma. Se ci dimenticassimo di chiamarlo verrebbe solamente creato un file vuoto, in quanto tale metodo permette di scaricare su file tutte le informazioni precedentemente bufferizzate. Prima di chiamare questo metodo va però chiamato il metodo *pieDiPagina* che ci permette l'inserimento delle istruzioni di dimensionamento della nostra finestra.

IL CUORE DEL PROGRAMMA

Passiamo alla spiegazione della classe *SwingCreator,* la quale, oltre a contenere il metodo *main,* è il fulcro intorno al quale si muove tutto il software.
Il costruttore di tale classe viene usato per costruire la GUI del programma. Per coerenza, le istruzioni sono state ricavate direttamente da questo programma! Ci sono chiaramente delle aggiunte che riguardano la gestione degli eventi come ad esempio è possibile selezionare il file *.frm* da convertire cliccando sul pulsante *Apri.* Questo ci aprirà un oggetto appartenente alla classe *JfileChooser* che ci permette di navigare all'interno del FileSystem del nostro computer. Per l'inserimento del logo di ioProgrammo si è ricorsi ad un piccolo trucco: si è creato un oggetto della classe *ImageIcon,* dopodichè lo si è associato ad una label senza testo. Esistono maniere più eleganti per l'inserimento di immagini in un Container, ma questo è sicuramente il più veloce. I due metodi principali di questa classe (i quali sono il cuore di tutto il programma) sono i seguenti:

```
private void iniziaConversione()
private void inserisciOggetto(String obj, String padre,
              String tipoPadre, BufferedReader br) throws
                                             IOException
```

Entrambi i metodi sono dichiarati *private,* questo perché non devono essere accessibili al di fuori di questa classe. Come vediamo, anche il metodo *inserisciOggetto* rimanda al chiamante la gestione dell'eccezione *IOException.* Sarà infatti compito del metodo *inizaConversione,* gestire tutti i problemi di input/output; questo fa si che il corpo del metodo sia racchiuso all'interno dei blocchi *try/catch.*
Il corpo di questo metodo è semplicissimo da comprendere: viene aperto il file *.frm* in lettura viene fatto ciclare una prima volta per verificare la quantità di oggetti totali al fine di dimensionare correttamente l'oggetto *Oggetto,* dopodichè viene eseguito un ulteriore ciclo, questa volta utile per il riconoscimento dei vari oggetti e l'inserimento degli stessi nel file Java. In pratica, ad ogni ciclo viene letta un'intera riga tramite il metodo *readLine* della classe *Buf-*

COMPILAZIONE

Per compilare i programmi scritti in java, i sorgenti devono essere salvati su disco con estensione *.java*. Accertatevi che tutte e tre le classi siano presenti nella medesima cartella, dopodichè, dalla finestra DOS, dopo esservi portati nella cartella ove risiedono i file, digitate il comando javac *SwingCreator.java*. Se non viene segnalato alcun errore significa che il programma è stato compilato correttamente. A questo punto per avviarlo digitate *java SwingCreator.*

feredReader, dopodichè viene utilizzato il metodo *indexOf* della classe *String* il quale accetta come parametro la stringa che vogliamo ricercare all'interno della stringa alla quale viene applicato il metodo e ritorna *–1* se non viene trovata, oppure l'indice alla quale tale sottostringa ha inizio. A questo metodo noi passeremo la stringa *Begin VB.* che indica l'inizio di un nuovo oggetto. Se il valore ritornato dal metodo sarà diverso da *–1* verrà chiamato il metodo *inserisciOggetto* che permette di estrapolare le varie informazioni riguardanti l'oggetto ed inserirle nell'oggetto *Oggetto.*
Il metodo si conclude creando un'istanza della classe *ScriviFile* che, come detto sopra, ci servirà per compilare il file in formato Java in quanto, a questo punto del programma, avremo tutte le informazioni che ci interessano contenute all'interno del nostro array di oggetti *Oggetto.*

NOTA

VARIABILI D'AMBIENTE

Per poter utilizzare i comandi *javac* e *java* da quasiasi posizione ci si trovi bisogna impostare la variabile d'ambiente PATH con il percorso corretto ove si trovano i programmi. Per accedere al pannello di modifica cliccare su *Start/Pannello di Controllo/Sistema/Avanzate/Variabili* d'ambiente. Si presenterà la schermata di Fig. 4. Da qui è possibile modificare o aggiungere variabili tramite i pulsanti *Nuovo* e *Modifica.*

COME CREARE I VARI OGGETTI

Ma il fulcro di tutto è il metodo *inserisciOggetto!* Tale metodo legge le righe dal file VB, estrapola le informazioni e, dopo aver creato un'istanza di *Oggetto,* compila le proprietà che saranno codificate in linguaggio Java dalla classe *ScriviFile.*
Vediamo come:

```
StringTokenizer token = new StringTokenizer(obj);
tipo = token.nextToken();
nome = token.nextToken();
oggetto[indice] = new Oggetto(tipo,nome);
padreTemp = oggetto[indice].getNome();
tipoPadreTemp = oggetto[indice].getTipo();
oggetto[indice].setPadre(padre);
oggetto[indice].setTipoPadre(tipoPadre);
```

A questo metodo viene passata la stringa letta dal file *(obj),* l'identificatore padre e *tipoPadre,* e l'oggetto *BufferedReader* per continuare a leggere nel file letto. Come primo passaggio viene creata un'istanza della classe *StringTokenizer* la quale ci permette di dividere una stringa in token, vale a dire parole separate da un carattere predefinito.
Nella sua forma più semplice, come carattere viene utilizzato lo spazio. Questo significa che la frase *Programmare* in Java viene divisa in tre token:

Programmare
in
Java

A noi risulta utile in quanto nella riga contenente la dicitura *Begin VB.* è presente anche il tipo di oggetto ed il nome assegnatogli in fase di progettazione. Tali parametri vengono passati al costruttore di *Oggetto* per creare correttamente l'istanza. Vengono assegnate anche le proprietà *padre* e *tipoPadre* che identificano l'oggetto nella gerarchia di scatole cinesi sopra esposta. Invece alle due variabili temporanee vengono assegnati i nomi dell'oggetto corrente.
Questo perché nel caso in cui nelle precedenti letture del file *.frm* non venisse riscontrata la chiusura dell'oggetto tramite la dicitura *End,* bensì nuovamente *Begin.VB,* significherebbe che il nuovo oggetto che andremo a creare sarà figlio dell'oggetto non ancora chiuso. Infatti per risolvere il problema degli oggetti annidati uno dentro l'altro, la funzione *inserisciOggetto* è stata costruita utilizzando la ricorsione: ogni riga dell'oggetto viene letta fino a quando non si incontra la dicitura *End.*
Ad ogni lettura vengono confrontate le proprietà lette con quelle riconosciute dal programma. In caso positivo, tali proprietà vengono inserite nell'oggetto *Oggetto.*
Se durante una nuova lettura viene riscontrata la dicitura *Begin VB.* viene nuovamente richiamato il metodo *inserisciOggetto,* e così fino alla fine del file.
Un altro metodo presente nella classe è *eliminaApici*:

```
private String eliminaApici(String stringa) {
  String temp = "";
  int idTemp = 0;
  for(int i = 0;i < stringa.length();i++)
     if(stringa.charAt(i) != '"')
        temp = temp + stringa.charAt(i);
     return temp;
}
```

Tale metodo accetta in ingresso una stringa e restituisce la stessa privata dei doppi apici. Questo risulta utile per le proprietà quali *Caption, Text, ToolTipText* il cui valore, all'interno del codice VB è compreso appunto tra apici.

CONCLUSIONI

Per questo mese basta così. Il programma proposto fino a questo punto è perfettamente funzionante, anche se permette la conversione di interfacce limitate. Il prossimo mese vedremo di estenderne le potenzialità come ad esempio il riconoscimento dei menù e l'inserimento di grafica quali linee, rettangoli, etc.
Per il momento, chi volesse, può effettuare le sue prove direttamente sui Form forniti insieme all'applicazione (per visionare i Form è necessario avere installato l'ambiente di sviluppo Visual Basic).
Buono studio e soprattutto buon divertimento nella creazione delle vostre GUI!
Alla prossima!

Alessandro Baldini

WordML e XPath per aggiornare feed RSS

Pubblicare via RSS

parte seconda

WordML è il nuovo formato standard di Microsoft Word. Impareremo a interpretarlo e a pubblicare su Internet i contenuti dei nostri documenti tramite il protocollo dei Blog: RSS

Nella scorsa puntata avevamo visto cosa è un feed RSS e come possiamo utilizzare .NET per produrre feed. Il nostro progetto era infatti quello di aggiungere ad una applicazione web la possibilità di produrre dei feed a partire dalla presenza di nuovi file in un repository di documenti. Utilizzando la caratteristica di salvataggio XML di Word 2003 potremo realizzare un servizio che faccia quanto ci serve. Utilizzeremo varie tecniche della programmazione OOP, tra cui il polimorfismo, l'ereditarietà e l'uso di interfacce. Capiremo come utilizzare i servizi del framework .NET per monitorare la modifica dei file in una cartella. Illustreremo come creare una architettura estensibile e aperta.
Vedremo come è strutturato un file WordML e come utilizzare XPath per estrarre informazioni. Tutto questo porterà allo sviluppo di una applicazione che produrrà automaticamente il file RSS aggiornato.

L'ARCHITETTURA DEL SISTEMA

Possiamo vedere l'architettura del sistema che andremo a realizzare sotto due punti di vista: la sequenza delle azioni che verranno prodotte, quindi una vista dinamica, e la struttura delle classi utilizzate, quindi una vista statica. In Fig. 1 vediamo un diagramma di stato che mostra gli stati principali in cui si trova il nostro servizio, in relazione alle operazioni che verranno effettuate. Queste operazioni sono schematizzate in Fig. 2. In sintesi:

1. il nostro servizio viene attivato;
2. un redattore produce un articolo in Word 2003;
3. l'articolo viene salvato in un repository, nel nostro caso una cartella;
4. il servizio si accorge che è stato aggiunto un nuovo file alla cartella, e quindi inizia il processo di estrazione delle informazioni dai file;
5. il servizio produce il file XML contenente il *feed RSS*;
6. il file viene pubblicato.

Per *repository* intendiamo un qualunque luogo "logico" in cui i redattori possono salvare i loro articoli WordML. Potremmo utilizzare un database, o un sito che espone un web service o anche un server FTP. Da un punto di vista statico, il nostro servizio avrà bisogno di alcuni componenti di base, come si evince dal diagramma delle sequenze della Fig. 3: una classe che realizzi il servizio; una classe che si occupi di sorvegliare il repository per informare il servizio di nuovi arrivi e per passare i documenti al servizio stesso; una classe che si occupi di gestire la produzione del file RSS, utilizzando fra le altre la classe RSS20 la cui struttura è già stata vista nella scorsa puntata; una classe estrattore, il cui scopo è quello di fare il parsing vero e proprio e quindi di estrarre le parti salienti dal documento WordML perché siano inserite nel feed RSS.

ESTENSIBILITÀ

Esaminando le considerazioni fatte nel paragrafo precedente ci rendiamo conto che sarebbe interessante progettare il sistema perché possa essere estensibile almeno in tre direzioni:

- Gestione di più tipi di repository.
- Gestione di più formati di file in input.
- Gestione di differenti strategie di individuazione ed estrazione delle parti interessanti dal testo.

Per realizzare quanto descritto non andremo a scrivere il nostro codice facendo diretto riferimento a classi, ma riferendoci ad interfacce. Definiremo due interfacce che esporranno tutte le caratteristiche comuni delle classi volute:
IRepositoryWatcher: verrà implementata dalla classe che farà comunicare la nostra applicazione con il repository. Suoi compiti saranno:

- sorvegliare il repository per avviare l'aggiornamento del file RSS qualora ci siano nuovi file o gli esistenti vengano modificati;
- fornire alla classe produttrice dell'RSS i conte-

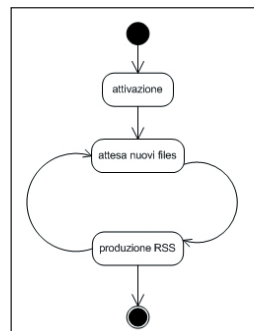



Fig. 1: Diagramma di stato del servizio

nuti dei file. In questo modo la vera natura del repository verrà nascosta al resto del programma. Così facendo, per esempio, grazie al polimorfismo la struttura del servizio non saprà se sta utilizzando la classe che prende i file dal filesystem o quella che invece li scarica da un sito FTP: saprà soltanto che sta utilizzando una classe che implementa l'interfaccia *IRepositoryWatcher.*

IContentExtractor: rappresenta la classe che estrae dal documento i contenuti che servono per produrre l'item RSS. Questa interfaccia è molto utile, in quanto potremmo progettare in seguito algoritmi più sofisticati di individuazione delle parti del testo. Se la classe che li realizzerà implementerà anche *IContentExtractor* potrà essere utilizzata senza modifiche al codice chiamante.

Classe	Responsabilità	Deriva da
Coordinatore	Si occupa di coordinare le varie classi. Dà l'inizio al processo di sorveglianza del repository, e chiama il produttore RSS assegnando il *ContentExtractor* corretto	
SimpleWordMLContent Extractor	Ha lo scopo di estrarre i dati da files in cui i dati siano salvati in un file WordML e contrassegnati da particolari stili	realizza l'interfaccia *IContentExtractor*
SimpleTextContent Extractor	Ha lo scopo di estrarre i dati da files in cui i dati siano disposti ognuno su una riga	realizza l'interfaccia *IContentExtractor*
ComplexTextContent Extractor	Ha lo scopo di estrarre i dati da files in cui i dati siano fra tags: per esempio <TITOLO> ciao </TITOLO> utilizza le regular expressions	realizza l'interfaccia *IContentExtractor*
WatcherCartella	Una IRepositoryWatcher che ha come repository una cartella di filesystem	realizza l'interfaccia *IRepositoryWatcher*
RSS20	La classe vista nello scorso numero, realizza il file RSS	
RSSProducer	Si occupa di produrre il file RSS tramite la classe RSS20	
FileDaRepository, DatiArticolo	Classi helper	

Tabella 1: Classi principali del programma

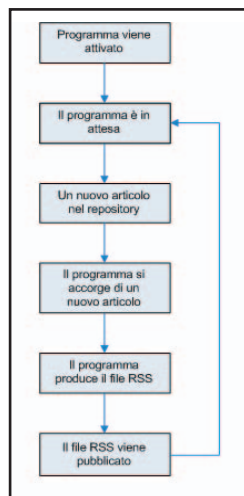



Fig. 2: ***Le operazioni effettuate dal servizio***

Le classi principali realizzate per il programma, e le loro responsabilità, sono riportante nella Tab. 1.
Nel CD e/o sul Web, troverete anche un file *.chm* con la documentazione completa delle classi, dei metodi e dei delegati realizzati.

CONTROLLARE IL REPOSITORY

La classe che implementa l'interfaccia *IRepositoryWatcher* deve poter informare il programma quando si ha un cambiamento nel repository. Nel nostro esempio il repository è il file system locale. Ci viene quindi in aiuto la classe del Framework .NET *FileSystemWatcher.* Questa classe fa proprio al caso nostro, infatti sorveglia una cartella, o un ramo di cartelle, e alza degli eventi in base ai cambiamenti. Utilizzeremo questa classe in modo asincrono all'interno della classe *WatcherCartella,* impostandone a true la proprietà *EnableRaisingEvents.* Resterà solo da registrare i nostri gestori agli eventi del *FileSystemWatcher* e il gioco sarà fatto. Nel corpo della classe sono presenti anche due metodi, *CominciaEnumerazioneFiles()* e *ProssimoFile()* che vengono utilizzati per scorrere e per estrarre i file dal repository. Ecco il codice della classe:

```
public class WatcherCartella: IRepositoryWatcher
{ //fields
  string _Path;
  FileSystemWatcher _FSW;
  FileInfo[] _Files;
  int _FileCorrente;
  /// <summary>
  /// costruttore
  /// </summary>
  /// <param name="Path">il path della cartella
                                    repository</param>
  public WatcherCartella(string Path)
  {  _Path=Path;
     _FSW= new System.IO.FileSystemWatcher(Path);
     _FSW.Filter ="*.*";
     _FSW.NotifyFilter =NotifyFilters.LastAccess |
                                NotifyFilters.LastWrite
     | NotifyFilters.FileName | NotifyFilters.DirectoryName;
     _FSW.Deleted+=new FileSystemEventHandler(
                                      _FSW_Deleted);
     _FSW.Created +=new FileSystemEventHandler(
                                      _FSW_Created);
     _FSW.Changed+=new FileSystemEventHandler(
                                   _FSW_Changed); }
  //per info su questi metodi si veda la dichiarazione
                                      di interfaccia
  #region IRepositoryWatcher Members
  public void CominciaAControllare()
  { _FSW.EnableRaisingEvents=true; }
  public void Smetti()
  { _FSW.EnableRaisingEvents=false;}
  public void CominciaEnumerazioneFiles()
  { DirectoryInfo DI= new DirectoryInfo (_Path);
  _Files =  DI.GetFiles();
  _FileCorrente=0; }
public FileDaRepository ProssimoFile()
{ ...}
```

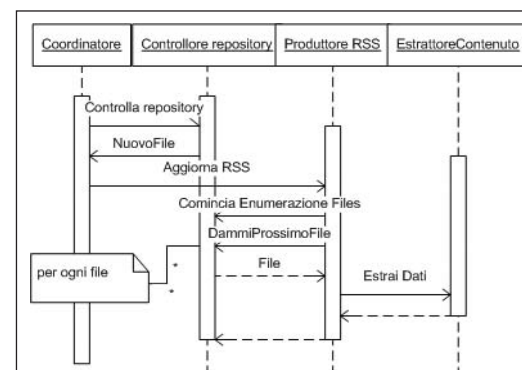



Fig. 3: ***Componenti di base del servizio***

WORDML: XML PER I DOCUMENTI

WordML, che sta per *WordprocessingML*, è un dialetto XML creato per descrivere in maniera aperta un documento, nel caso specifico un documento Word. Il fatto che sia XML ci porta alle seguenti considerazioni:

- Possiamo utilizzare gli strumenti XML del framework .NET per fare il parsing del documento.
- Le nostre considerazioni varranno anche per documenti non Word, purché conformi allo schema WordML.
- Sarebbe interessante che il nostro sistema fosse espandibile, per poter implementare migliori strategie di rilevamento delle parti del testo.

Per poter esaminare i file XML che ci verranno sottoposti, occorre conoscere lo schema e il significato degli elementi che compongono un file WordML.
Lo schema di WordML è piuttosto complesso: introdurremo qui le caratteristiche di base che ci basteranno per un esame rozzo ma efficace di documenti ben strutturati. Nei riferimenti riportati a lato troverete una documentazione più dettagliata ed utile ad utilizzi professionali. I principali elementi di alto livello in un documento WordML sono l'elemento *root: wordDocument* e gli elementi figlio indicati nella *Tabella 2*. Infine, il vero e proprio corpo del documento, l'elemento *body*: il contenuto del documento. All'interno di questo, una serie di elementi, che vanno a definire le sezioni, gli stili, e i paragrafi, tutti elencati nella *Tabella 3*. Quindi il più semplice documento WordML potrebbe essere:

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
   xmlns:w=http://schemas.microsoft.com/office/word
      /2003/wordml xmlns:o="urn:schemas-microsoft-com:
                                          office:office">
<o:DocumentProperties>
  <o:Title>Il primo articolo</o:Title>
  <o:Author>Marco Poponi</o:Author>
</o:DocumentProperties>
<w:body>  <w:p><w:r>
        <w:t>Il mio articolo</w:t>
      </w:r> </w:p>
</w:body>
</w:wordDocument
```

In cui è presente solo un pagarafo di stile *Normale* (quindi senza specifiche di stile) e con testo: Il mio articolo. Si noti la processing instruction *<?mso-application progid="Word.Document"?>* che istruisce Internet Explorer a non aprire normalmente il file XML ma a passarlo a Word. È opportuno notare che queste informazioni non servono solo per leggere un file WordML prodotto da Word, ma possono essere utilizzate anche per produrre un file che Word può leggere.

L'ESTRAZIONE

La complessità e l'efficacia dell'algoritmo di estrazione del contenuto del file dipendono in gran parte da due fattori:

1. quanto e come è strutturato il file di partenza;
2. quali informazioni devono essere estratte.

Nel nostro esempio supporremo che il redattore faccia un uso avveduto degli stili, ovvero che assegni uno stile conosciuto e preciso ad ogni parte dell'articolo. Per esempio, il titolo sarà realizzato nello stile *TitoloPrincipale*, mentre il riassunto sarà, appunto, realizzato nello stile *Riassunto* e così via. In Fig. 5 è riportato lo schema di massima dell'algoritmo utilizzato. L'estrazione delle informazioni viene fatta attraverso l'uso di query XPath. Per quanto riguarda l'estrazione delle informazioni presenti sul nodo *DocumentProperties* sarà una query sul nome del nodo, per quanto riguarda invece i dati nel corpo del documento la query sarà più complessa, e farà riferimento al nodo di stile figlio del nodo paragrafo. Il codice della classe che estrae i dati WordML sarà (in neretto le query XPath):

```
public class SimpleWordMLContentExtractor:IContentExtractor
{ private XmlDocument XDoc;
/// <summary>
/// Metodo che utilizza una query XPath per estrarre
                              tutto il testo dell'articolo
/// che ha un certo stile passato come param
/// </summary>
/// <param name="Style">lo stile da usare per la
                                             query</param>
/// <returns></returns>
private string EstraiDato(string Style)
{  //devo definire un XmlNamespaceManager per
                          gestire i namespaces nella query
   XmlNamespaceManager XMLNS= new
               XmlNamespaceManager(XDoc.NameTable);
   XMLNS.AddNamespace("o","
               urn:schemas-microsoft-com:office:office");
   XMLNS.AddNamespace("w","http://schemas.
               microsoft.com/office/word/2003/wordml");
   //la query XPath
   string
   XQuery=@"/w:wordDocument/w:body//*/w:p[
        descendant::*[@w:val='STILE']]//*/w:t";
   XQuery= XQuery.Replace("STILE",Style);
   XmlNodeList Nodelist = (XDoc.SelectNodes(
                                         XQuery,XMLNS));
```

- **Elemento documentProperties**: qui sono presenti utilissime informazioni sul documento, quali autore, data ecc.
- **Elemento fonts**: informazioni sui font.
- **Elemento list**: definizioni di stili di lista.
- **Elemento styles**: definizioni degli stili utilizzati nel documento.
- **Elemento docPr**: opzioni generali del documento.

Tabella 2: Gli elementi figlio di wordDocument

- **Elemento p**: un paragrafo
- **Elemento r (run)**: un insieme contiguo di elementi WordML omogenei per proprietà
- **Elemento t**: un brano di testo
- **Elemento pStyle**: inizio di uno stile.

Tabella 3: Gli elementi che compongono il body

```
  //ciclo fra tutti i nodi trovati ed estraggo il testo
  string ret="";
  if (Nodelist.Count!=0)
  { foreach(XmlNode Nodo in Nodelist)
      { ret+=(Nodo as XmlElement).InnerText ;}
  } return ret; }
#region IContentExtractor Members
/// <summary>
/// estrae i dati, utilizzando query XPath
/// </summary>
/// <param name="FileDaEstrarre"></param>
/// <returns></returns>
public DatiArticolo EstraiArticolo(FileDaRepository
                                        FileDaEstrarre)
{ if (FileDaEstrarre.Nome.EndsWith(".xml"))
  { //carico il documento in un doc XML
      ...
      ///utilizzo una semplice query XPath per estrarre i dati
      Dati.Autore=(XDoc.SelectSingleNode("//o:Author",
                  XMLNS) as XmlElement ).InnerText ;
      Dati.DataPubblicazione=(XDoc.SelectSingleNode(
       "//o:Created",XMLNS) as XmlElement ).InnerText ;
      ///richiamo una query XPath più complessa per
                       estrarre i dati nel corpo del testo
      Dati.Titolo=EstraiDato("TitoloArticolo");
      Dati.URLDiPubblicazione=EstraiDato(
                                   "URLPubblicazione");
      Dati.Riassunto=EstraiDato("Riassunto");
      return Dati; }
  else
  { return null; } } #endregion }
```

ESTENSIBILITÀ

Abbiamo deciso di realizzare una applicazione Windows Forms (e non un servizio, come sarebbe stato più ortodosso se il progetto fosse andato in produzione) per permettervi di sperimentare quanto abbiamo visto in pratica, specialmente l'estensibilità. L'interfaccia grafica dell'applicazione (vedi Fig. 6) permette di specificare quale sia il repository (la cartella nel nostro caso) da utilizzare, il path in cui verrà salvato il file RSS prodotto e soprattutto quale classe *IContentExtractor* verrà passata al Coordinatore. Per mostrare in maggiore dettaglio il meccanismo di estensibilità adottato sono state scritte altre due classi che estraggono il contenuto di un articolo oltre a quella per il formato WordML. La prima, *SimpleTextContentExtractor*, assume che le varie informazioni siano presenti in un file di testo su righe diverse. La seconda, *ComplexTextContentExtractor*, si aspetta, invece, un file *.txtc* un po' più strutturato, in cui i dati utili siano scritti fra tag XML (esempio: *<TITOLO> questo è il titolo</TITOLO>*) ed utilizza Regular Expressions per operare il parsing. Nessuna delle classi che realizzano il programma è al corrente a quale classe appartenga l'oggetto derivante da *IContentExtractor* che verrà utilizzato. Nel nostro esempio è l'interfaccia grafica che sceglie quale classe implementare e passare al Coordinatore a seconda dell'input dell'utente. Questo permette, qualora si individui un algoritmo più raffinato, o si cambi il formato dei file, di aggiornare l'intero programma semplicemente scrivendo una nuova classe: tutto il resto continuerà a funzionare

GENERATORE DI RSS

Abbiamo visto nello scorso numero come scrivere una classe che realizzi facilmente un file RSS. Nel nostro progetto questa classe verrà utilizzata nel modo che conosciamo dalla classe *RSSProducer. RSSProducer* utilizzerà inoltre i servizi della *IRepositoryWatcher* per enumerare ed estrarre i file dal repository:

```
_RWatcher.CominciaEnumerazioneFiles();
FileDaRepository FDR= _RWatcher.ProssimoFile();
while(FDR!=null)
{  (...) aggiunge l'item
   FDR= _RWatcher.ProssimoFile(); }
```

E poi salverà il file in un modo un po' diverso da quanto visto nel numero scorso, infatti utilizza *XmlTextWriter* che ci permette di specificare l'encoding per il file XML generato. Lo vediamo nel seguente brano di codice, in cui RSS è l'oggetto che realizza il file RSS:

```
XmlSerializer Ser= new XmlSerializer(typeof(rssClass));
System.Xml.XmlTextWriter XMLTW= new System.Xml.
    XmlTextWriter(path , System.Text.Encoding.UTF8);
Ser.Serialize(XMLTW,RSS);
XMLTW.Close();
```

CONCLUSIONI

Abbiamo quindi realizzato tutti gli elementi che ci servono per la costruzione del nostro progetto. Non ci resta che "assemblarli" in maniera che il tutto funzioni. Il cuore del progetto sarà la classe Coordinatore che, appunto, coordinerà tutte le altre classi. Analizziamo tre metodi di questa classe:

1. Il costruttore, che prende come parametro la *IContentExtractor* da utilizzare.
2. Il metodo che dà inizio al processo.
3. Il metodo che viene richiamato quando si ha un nuovo file e si occupa di aggiornare il file RSS.

Il risultato lo trovate nella classe costruttore, presente nel CD allegato alla rivista.

Marco Poponi

http://msdn.microsoft.com/office/understanding/xmloffice/tools/default.aspx
il centro msdn di download degli strumenti su XML in Office. Consiglio fortemente il primo download, Office 2003 XML Reference Schemas in cui è possibile trovare una completa documentazione su WordML.

http://msdn.microsoft.com/office/understanding/xmloffice/default.aspx
il centro msdn sull'uso di XML in Office

http://sourceforge.net/projects/xpe/
XPath Explorer, un semplice ma potente programma Java Open Source che permette di provare query XPath su file XML.

I FILE NEL CD

Nel CD e sul sito Web allegato alla rivista è presente il codice sorgente dell'aplicazione completa descritta in questo articolo, dei file di esempio e un file di documentazione completo delle classi presenti nell'applicazione.

Lista variabile di argomenti e altre innovazioni

Le novità di Java 1.5

parte seconda

Continuiamo nel nostro approfondimento sulle novità di Tiger, affrontando *Metadati* e *Import Statici*: non esattamente aspetti chiave del linguaggio, ma che possono rivelarsi di grande utilità

Cominciamo questa nuova puntata con un esempio che non potrà lasciarvi indifferenti: secondo la nuova sintassi di Java 1.5, la seguente riga di codice:

```
public void test(Object ... args)
```

dichiara un metodo che può accettare in input un numero variabile di parametri (argomenti) di tipo *Object*. Come si può notare, deve essere specificato il tipo, tre puntini ed il nome del parametro. Il metodo potrà quindi essere invocato passando un numero arbitrario di oggetti:

```
test(obj);
test(obj1, obj2);
test(obj1, obj2, obj3);
test(obj1, obj2, obj3, obj4);
```

e persino nessun parametro:

```
test();
```

All'interno del metodo, gli argomenti passati saranno visti come elementi di un array. Nel seguente esempio mostreremo un metodo che accetta in input una lista variabile di oggetti, ne visualizza la dimensione e, in seguito, ogni singolo elemento:

```
public void test(Object ... args) {
  System.out.println
   ("Numero di parametri: " + args.length);
  for(Object obj: args) {
    System.out.println(obj); }
}
```

Invocando il metodo *test* e passando la stringa *"Hello"* in input, otterremo l'output di Fig. 1:

```
test("hello");
```

In questo secondo esempio, passeremo due stringe, una data e un numero. L'output lo trovate in Fig. 2

```
test("hello","ciao",new Date(), 3);
```

Resta inteso che è possibile anche invocare *test()* senza alcun parametro:

```
test();
```

La lista di parametri potrà appartenere a qualsiasi tipo valido. Nel seguente esempio, il metodo riceverà in input una lista di interi:

```
public void testInt(Integer ... ints) {
    System.out.println
     ("Numero di parametri: " + ints.length);
     for(Integer i: ints) {System.out.println(i);}
}
```

Ovviamente in input saranno accettati solo parametri interi:

```
testInt();
testInt(3);
testInt(5,8);
testInt(27,-9,72);
```

Se proviamo ad invocare *testInt* passando un tipo non intero (per esempio una stringa):

```
testInt(5,"otto"); // Errato!
```

otterremo un errore in compilazione.
Un metodo che ha una lista variabile di argomenti, può avere in input anche altri parametri; di seguito

Fig. 1: Il primo esperimento

Fig. 2: Ancora un conteggio di parametri

ne portiamo un esempio:

```
public void test2(String str,
  Integer ... ints) {
    System.out.println("str = " + str);
    System.out.println
      ("Numero di parametri: " + ints.length);
    for(Integer i: ints) { System.out.println(i); }}
```

Il primo parametro del metodo *test2* è una stringa, il secondo è invece una lista variabile di argomenti interi. Un esempio d'invocazione potrebbe essere il seguente:

```
testInt2("Hello",6,8);
```

In questo caso il primo parametro è obbligatorio; non sarà quindi consentito invocare il metodo *test2* senza parametri:

```
testInt2(); // Non consentito!
```

Quando un metodo presenta argomenti normali e una lista variabile di argomenti, questa, al fine di evitare ambiguità, deve essere sempre specificata come ultimo parametro. Detto ciò, la seguente definizione non è consentita:

```
// Dichiarazione non consentita!
public void test3(Integer ... ints, String str) { ... }
```

Di conseguenza, non è consentito avere metodi con più di una lista variabile di argomenti:

```
// Dichiarazione non consentita!
public void test4(Integer ... ints1, Integer ints2) { ... }
```

poiché entrambe dovrebbero essere definite come ultimo parametro e ciò è impossibile.

UN ESEMPIO

L'esempio classico relativo alle funzioni con numero variabile di argomenti è il metodo *printf* che sarà esaminato in un prossimo articolo. Esso riceve in input una stringa speciale, detta *di formattazione*, ed una lista variabile di elementi che verranno scritti sullo stream di output. Un esempio di invocazione:

```
String str = "Ciao";
int e = 29;
System.out.printf
  ("%s, sono Mario ed ho %d anni", str, e);
```

Un altro contesto dove i metodi a lista variabile d'argomenti possono risultare utili è quello relativo alle strutture dati classiche. Per esempio, potremmo definire la primitiva *add* della struttura dati coda mediante il metodo *add* che ha in input una lista variabile d'argomenti:

```
public void add(Object ... elements) { ... }
```

In questo modo sarà possibile, usando lo stesso metodo, aggiungere anche più di un elemento per volta:

```
add("Mario");
add("Valentina","Luisa");
```

ANNOTAZIONI

I metadati sono informazioni aggiuntive che possono essere associate a classi, interfacce, metodi e singole variabili. Per esempio, si potrebbero creare dei tool che leggano (e successivamente elaborino) metadati dal codice sorgente, dal codice compilato oppure a run-time accedendo direttamente alla Virtual Machine. Java 1.5 introduce il concetto di annotazione *(annotation)* per definire metadati. Una annotazione è un'informazione aggiuntiva che precede la definizione di un'entità Java. Ecco un esempio:

```
@SpecialMethod public void test() { ... }
```

L'annotazione usata in questo esempio è *@SpecialMethod*. Ma qual è il suo significato? Indica che qualche tool avrà bisogno di quest'informazione aggiuntiva alla classe. Per esempio, potremmo crearne uno che elenca tutti i metodi annotati da *@SpecialMethod*. Per quale motivo? Dipende dall'obiettivo del tool. Consideriamo quest'altra annotazione:

```
@Autore("Mario Rossi") public void test() { ... }
```

In base a questa nuova informazione, potremmo costruire un tool che individui tutti i metodi annotati con *@Autore* e ne visualizzi il nome. I tool, come abbiamo accennato, possono operare sul codice sorgente, sul codice compilato o a run-time. Partendo da ciò, ed utilizzando l'annotazione *@Autore*, potremmo implementarne tre tipi differenti che:

1. a partire dal codice sorgente, individui l'autore per ogni metodo;
2. a partire dal bytecode, individui l'autore per ogni metodo;
3. a partire da un'applicazione in esecuzione, individui l'autore per ogni metodo.

Utilizzare la strategia adatta dipende al contesto in cui il tool deve operare. Per esempio, un programma che generi automaticamente documentazione a partire dalle annotazioni (tipo *JavaDoc*), adotterà la prima strategia.

NOTA

PERCHÉ IMPORT "STATIC"?

Qualcuno si sarà certamente chiesto il motivo per cui è stata introdotta la notazione *import static* e non si è utilizzato solamente *import*, visto che comunque non avrebbe generato nessuna ambiguità. Le ragioni sono fondamentalmente due:

1. Per distinguere a colpo d'occhio gli import relativi alle classi e alle interfacce da quelli relativi ai membri e ai metodi statici.
2. Per ricordare al programmatore che solo membri e metodi statici possono essere importati. (Infatti se si cerca di importare un membro non statico il compilatore segnalerà un errore).

Su questo secondo punto sono particolarmente d'accordo, poiché utilizzando import (senza static) per questo nuovo tipo di funzionalità avrebbe generato molta confusione.

DEFINIZIONE DI UNA ANNOTAZIONE

Nel paragrafo precedente abbiamo visto come annotare un'entità Java, ma non come e dove sia stata definita l'annotazione. L'annotazione *@SpecialMethod* può essere definita come una interfaccia speciale nel seguente modo:

```
@interface SpecialMethod  {}
```

Questa interfaccia, rappresenta un tipo particolare di annotazione denominata marcatore, la sua particolarità è di non definire nè metodi nè informazioni aggiuntive, per questa sua caratteristica può solo marcare l'entità Java che la seguirà. L'annotazione @Autore invece dovrà essere definita come segue:

```
@interface Autore  {String nome();}
```

In questo caso, l'annotazione *@Autore* definisce il metodo *nome*. Questo tipo di annotazione è chiamata a membro singolo, poiché definisce un solo metodo. Quindi con:

```
@Autore("Mario Rossi") ...
```

si utilizza l'annotazione *@Autore* in cui il metodo nome restituisce le stringa *"Mario Rossi"*. I marcatori e le annotazioni a membro singolo sono un caso particolare delle annotazioni dette normali, ovvero che contengono più metodi. Ad esempio:

```
@interface Info {
  int codice();
  String autore(); }
```

L'annotazione *@Info* sarà utilizzata nel seguente modo:

```
@Info(codice=100, autore="Mario Rossi") ...
```

Esistono altri tipi più complessi di annotazione (di array, complesse, con valori di default, ecc).

L'ANNOTAZIONE @OVERRIDES

Java 1.5 definisce alcune annotazioni standard. Quella più utile è certamente l'annotazione *@Overrides*. Essa può essere utilizzata per specificare esplicitamente che un metodo effettua l'overriding dell'omonimo presente nella classe base. Il compilatore userà l'annotazione per controllare se l'overriding è corretto. Supponiamo di avere la classe *A:*

```
class A {
  public void foo() { ... } }
```

e la classe *B* che effettua l'overriding del metodo *foo*, ma in modo errato:

```
class B extends A {
  public void foo(int n) { ...  } }
```

Bene, ciò che abbiamo fatto è semplicemente l'overloading del metodo foo e non l'overriding. Il compilatore non segnalerà nessun errore, poiché non conosce le nostre intenzioni. Se invece utilizziamo l'annotazione *@Overrides*:

```
class B extends A {
  // Errore in compilazione!
  @Overrides public void foo(int n) {...} }
```

allora il compilatore segnalerà l'errore e noi ci accorgeremo di aver utilizzato una signature diversa.
La definizione corretta sarà:

```
class B extends A {
  @Overrides public void foo() {... } }
```

Un tipico esempio di errore di questo tipo è quando si vuole ridefinire il metodo *equals* per una nostra classe (per esempio la classe *Persona*).
Spesso si è tentati di scrivere:

```
public boolean equals(Persona p){...}
```

invece di:

```
public boolean equals(Object p){...}
```

che è la versione corretta.
Se utilizziamo l'annotazione *@Overrides* il compilatore ci informerà dell'eventuale errore:

```
//Errore in compilazione
@Overrides public boolean equals(Persona p){...}
//Definizione corretta
@Overrides public boolean equals(Object p){...}
```

LE ANNOTAZIONI A RUN-TIME

Supponiamo di voler creare un tool che legga le annotazioni a run-time. La prima operazione da fare è quella di comunicare che un'annotazione può essere letta a run-time. Questo può essere fatto mediante l'annotazione standard *@Retention* nel seguente modo:

```
import static
java.lang.annotation.RetentionPolicy.*;
```

SUL WEB

Sun Microsystems, JavaTM 2 SDK, Standard Edition, Version 1.5.0 - Summary of New Features and Enhancements.
http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html

Calvin Austin, J2SE 1.5 in a Nutshell.
http://java.sun.com/developer/technicalArticles/releases/j2se15/

Sun Microsystems, JSR 201: Extending the JavaTM Programming Language with Enumerations, Autoboxing, Enhanced for loops and Static Import.
http://jcp.org/en/jsr/detail?id=201

Sun Microsystems, Importing Static Members in the JavaTM Programming Language
http://jcp.org/aboutJava/communityprocess/jsr/tiger/static-import.html

Sun Microsystems, JSR 175: A Metadata Facility for the JavaTM Programming Language
http://jcp.org/en/jsr/detail?id=175

```
@Retention(RUNTIME) @interface Info {
  int codice();
  String autore();}
```

Abbiamo quindi definito l'annotazione Info, avente i metodi codice e autore, che a sua volta è annotata da *@Retention(RUNTIME)*. Siamo in presenza di un'annotazione ad un'annotazione. La costante *RUNTIME*, definita nell'enumerazione *RetentionPolicy*, specifica che le informazioni presenti dell'annotazione devono essere disponibili a run-time.
Altri valori possono essere:

- **SOURCE** - L'annotazione sarà visibile solo a livello di codice sorgente
- **CLASS** *(default)* - L'annotazione sarà visibile solo a livello di codice compilato

Nel seguente esempio utilizzeremo l'annotazione *@Info* sia per una classe, sia per un metodo:

```
@Info(codice=100, autore="Mario Rossi")
public class Persona {
  ...
  @Info(codice=101, autore="Antonio Bianchi")
  public void test() { ...}
}
```

***Fig. 3:** L'annotazione associata alla classe Persona*

Noterete che la classe *Persona* è annotata da *@Info*, dove il codice è *100* e l'autore è *Mario Rossi*. Il metodo test invece è annotato da *@Info*, dove il codice è *101* e l'autore è *Antonio Bianchi*.
Per accedere all'annotazione @Info, relativa alla classe *Persona*, si può procedere nel seguente modo:

```
Persona p = new Persona();
Info info = p.getClass().getAnnotation(Info.class);
System.out.println("Codice = " + info.codice());
System.out.println("Autore = " + info.autore());
```

Il metodo *getAnnotation* ha in input il tipo dell'annotazione *(Info)* e restituisce l'istanza dell'annotazione associata alla classe *Persona* (Fig. 3).
Analogamente si può operare con l'annotazione associata al metodo test:

```
Persona p = new Persona();
Info info = p.getClass().getMethod("test")
  .getAnnotation(Info.class);
System.out.println("Codice = " + info.codice());
System.out.println("Autore = " + info.autore());
```

***Fig. 4:** L'annotazione del metodo Get*

In questo caso il nostro output sarà come in Fig. 4.
Consideriamo quest'altro esempio:

```
@SpecialMethod
@Info(codice=101, autore="Antonio Bianchi")
public void test() {... }
```

In questo caso il metodo *test* ha due annotazioni: *@Info* e *@SpecialMethod*. Per ottenere a run-time l'elenco di tutte le annotazioni associate al metodo *test*, si deve procedere nel seguente modo.

```
Persona p = new Persona();
Annotation a = p.getClass()
  .getMethod("test")
  .getAnnotations();
for (int i=0; i<a.length ; i++) {
  System.out.println
    ("a["+i+"]="+a[i]+" "); }
```

Come si può vedere, il metodo *getAnnotations* restituisce tutte le annotazioni per il metodo test come array di oggetti di tipo *Annotation*, ovvero la classe base per ogni annotazione.

IMPORTARE DATI MEMBRO STATICI

I dati membro statici vengono spesso utilizzati per definire delle costanti, che in seguito possono essere riferite ricorrendo alla notazione:

```
NomeClasse.IDENTIFICATORE
```

Per esempio, la classe *BorderLayout* definisce la costante *CENTER*. Per utilizzarla è necessario importare questa classe (o tutto il package che la contiene) e riferire la costante mediante la notazione *BorderLayout.CENTER*, come si può vedere dal seguente esempio:

```
import javax.swing.*;
import java.awt.*;
...
JFrame f = new JFrame();
f.getContentPane().add(
  new JPanel(), BorderLayout.CENTER);
```

Con i nuovi import statici sarà possibile evitare di specificare il nome della classe quando si vuole riferire un valore statico. Se all'inizio del nostro file ci sarà il comando di *import*:

```
import static java.awt.BorderLayout.CENTER;
```

allora non sarà più necessario specificare il nome della classe, ma solo quello della costante, come mostrato dal seguente esempio:

```
import javax.swing.*;
import java.awt.*;
import static java.awt.BorderLayout.CENTER;
...
JFrame f = new JFrame();
```

```
f.getContentPane().add(new JPanel(), CENTER);
```

Utilizzando la *wild-card* * sarà possibile importare tutti i membri statici relativi ad una classe. Consideriamo a tale proposito il seguente esempio:

```
import javax.swing.*;
import java.awt.*;
import static java.awt.BorderLayout.*;
...
JFrame f = new JFrame();
f.getContentPane().add(new JPanel(), CENTER);
f.getContentPane().add(new JPanel(), EAST);
```

Avendo utilizzato:

```
import static java.awt.BorderLayout.*;
```

è stato possibile riferire sia *CENTER*, sia *EAST* senza specificare il nome della classe *BorderLayout*.

IMPORTARE METODI STATICI

Come abbiamo già accennato, l'import statico può essere applicato anche ai metodi statici. Il modo di operare è pressoché identico a quello visto per i dati membro statici. Se la classe *test.MiaClasse* contiene il metodo statico *foo*, un'applicazione potrà invocarlo semplicemente utilizzando l'import statico in questo modo:

```
import static test.MiaClasse.*;
...
foo();
```

Un contesto dove importare metodi statici sarà molto utile è quello relativo alle funzioni matematiche presenti in *java.lang.Math*, che sono metodi statici. Utilizzando l'import statico possiamo evitare di specificare la classe *Math* ad ogni invocazione.

Per esempio, la chiamata:

```
double r = Math.sqrt(25);
```

diventa:

```
import static java.lang.Math.*;
...
double r = sqrt(25);
```

In generale, le chiamate:

```
Math.abs(x)
Math.sqrt(x)
Math.max(a, b)
```

diventeranno:

```
abs(x)
sqrt(x)
max(a, b)
```

Sebbene questo modo di operare sia indubbiamente comodo (ci siamo sbarazzati del nome della classe), a mio avvio rappresenta un passo indietro dal punto di vista della programmazione orientata agli oggetti. Infatti, nessuno ci vieta di creare decine di metodi statici in una singola classe e di invocarli in seguito, nello stesso modo in cui si invocano le funzioni del C o del Pascal. Ma c'è di peggio: medianti gli import statici, se ci pensate bene, è possibile di fatto creare qualcosa di moto simile alle variabili globali!

AMBIGUITÀ

Al fine di evitare ambiguità, il compilatore di Java 1.5 non consente di importare staticamente, nello stesso file, due dati membro aventi lo stesso nome, ma appartenenti a classi diverse. Supponiamo che la classe *A* e la classe *B* definiscano entrambe la costante *SEPARATOR*:

```
package test;
public class A {
  public final static char SEPARATOR = ';';}
package test;
public class B {
  public final static char SEPARATOR = ':';
}
```

Importiamo ora i membri statici delle due classi:

```
import static test.A.*;
import static test.B.*;
...
System.out.println(SEPARATOR);
```

Compilando otterremo il seguente errore:

```
Esempio.java:7: reference to SEPARATOR is ambiguous,
          both variable SEPARATOR in test.A and variable
         SEPARATOR in test.B match System.out.println(
                                           SEPARATOR);
```

Siamo in presenza di un'ambiguità: il compilatore non sa quale costante *SEPARATOR* utilizzare, quella presente in *A* oppure quella presente in *B*.
In questi casi sarà necessario ricorrere al nome della classe, quindi *A.SEPARATOR* per indicare la costante appartenente alla classe *A* e *B.SEPARATOR* per indicare la costante appartenente alla classe *B*.

Giuseppe Naccarato

NOTA

IMPORT STATICI

Come tutti sappiamo, affinché una classe o un'interfaccia possa essere utilizzata all'interno di un file, deve essere importata attraverso la parola chiave *import* (tale operazione non è necessaria se la classe o l'interfaccia appartengono allo stesso package del file). Ecco due banalissimi esempi di import:

```
import java.util.ArraList;
import java.io.*;
```

Con Java 1.5 è possibile importare singoli membri statici (dati o metodi) utilizzando la notazione *import static*.

Parsing del Web come fonte di informazioni

Stock Spy

parte seconda

Stock Spy è un'applicazione 100% Java che, connettendosi ad Internet, preleva le quotazioni di borsa. In questo secondo appuntamento definiremo l'interfaccia dell'applicazione utilizzando Swing

Il mese scorso abbiamo visto come prelevare da Internet, in tempo reale, le quotazioni dei titoli relativi alla borsa italiana. La classe *StockConnector* che abbiamo implementato effettua una connessione ad un provider di dati e, grazie allo *StockParser*, estrae quotazioni ed informazioni relative a tutti i titoli del Mibtel. Questo mese vedremo come utilizzare *StockConnector* per realizzare un'applicazione, basata su Swing, che visualizza in tempo reale le quotazioni dei titoli del Mib30. Prima di iniziare spendiamo qualche parola su *StockConnector*.

L'INTERFACCIA STOCKCONNECTOR

L'implementazione dell'interfaccia *StockConnector*, come abbiamo visto, dipende dal provider dei dati. Per ogni provider è necessario un'interfaccia diversa. Lo stesso discorso vale per *StockParser*. Come ricorderete, l'implementazione proposta nell'articolo precedente usa come provider il sito Kataweb Finanza. Abbiamo quindi implementato *KatawebConnector* e *KatawebParser* in modo che si adattino alle caratteristiche del sito, ovvero le varie URL per prelevare i dati, il metodo Http (GET) e il formato html in output.
Sebbene un'applicazione possa istanziare direttamente *KatawebConnector,* ciò non rappresenta una buona strategia. Infatti, se il sito scelto come provider non dovesse più fornire il servizio, la nostra applicazione diventerebbe inutilizzabile. Un approccio più intelligente è quello di caricare l'implementazione a run-time, utilizzando il metodo *Class.forName*, memorizzando il nome della classe in un file di proprietà. In questo modo, se il nostro provider non dovesse più essere attivo, potremmo implementare *StockConnector* basandoci su un altro provider e specificare questa nuova classe nel file di proprietà. Dato che l'applicazione carica a run-time l'implementazione ed essa rispetta l'interfaccia *StockConnector*, tutto continuerà a funzionare come prima. Implementiamo adesso una semplice applicazione che invochi i metodi di *StockConnector* basandoci sulla strategia appena descritta. Il primo passo è quello di creare un file di proprietà *(stockspy.properties)* e specificare il nome dell'implementazione di *StockConnector* nella proprietà *stock.connector.class:*

```
#stockspy.properties
stock.connector.class=KatawebConnector
```

L'applicazione leggerà quindi il valore di *stock.connector.class* e lo utilizzerà per creare un'istanza di *StockConnector* basato sull'implementazione *KatawebConnector:*

```
Properties properties = new Properties(
                              System.getProperties());
properties.load(new FileInputStream("stockspy.properties"));
String connectorName = properties.getProperty(
                              "stock.connector.class");
StockConnector connector = (StockConnector)
        Class.forName(connectorName).newInstance();
connector.connect(properties);
```

Se in futuro si dovrà cambiare provider, sarà sufficiente implementare *StockConnector* e specificare il nome della nuova classe nella proprietà *stock.connector.class*. Se, ad esempio, il nuovo provider è *Borsa Italia* e il nome della classe *BorsaItaliaConnector,* il file di proprietà diventerà il seguente:

```
#stockspy.properties
stock.connector.class=BorsaItaliaConnector
```

Dopo aver istanziato *StockConnector* mediante il metodo *Class.forName,* sarà possibile utilizzare l'istanza *connector* ed invocarne i metodi. Nell'esempio che segue verranno visualizzate tutte quotazione delle azioni che iniziano con la lettera *'t':*

```
StockQuote q[] = connector.getByLetter('t');
for (int i = 0; i < q.length; i++) {System.out.println(q[i]); }
```

il cui output potete apprezzare in Fig. 1.

AVVIO DA UNA RETE CON PROXY

Eseguendo Stock Spy all'interno di una rete con proxy, sarà necessario specificare le proprietà di sistema *proxySet, proxyHost* e *proxyPort.* L'impostazione può essere effettuata specificano l'opzione *-D* della Java Virtual Machine oppure inserendo le proprietà nel file *stockspy.properties,* come di seguito:

```
proxySet=true
proxyHost=myproxy.com
proxyPort=3128
```

dove *myproxy.com* sarà il nome del vostro proxy e *3128* la porta. La vostra applicazione avrà il compito di leggere le proprietà ed assegnarle alle omonime proprietà di sistema. In *Stock Spy,* questo compito sarà effettuato nel programma principale dal metodo statico *loadProperties:*

```
private static Properties loadProperties() throws Exception {
  Properties properties = new Properties(
                                       System.getProperties());
  properties.load(new FileInputStream(
                                       "stockspy.properties"));
  String sProxySet = properties.getProperty("proxySet");
  if (sProxySet!=null && sProxySet.equalsIgnoreCase(
                                                   "true")) {
    System.getProperties().setProperty("proxySet",sProxySet);
    String sProxyHost = properties.getProperty("proxyHost");
    if (sProxyHost!= null)
      System.getProperties().setProperty(
                                    "proxyHost",sProxyHost);
    String sProxyPort = properties.getProperty("proxyPort");
    if (sProxyPort!= null)
      System.getProperties().setProperty(
                                    "proxyPort",sProxyPort);}
  return properties; }
```

Come si può notare, il metodo legge tutte le proprietà dal file *stockspy.properties* e le restituisce.
Inoltre, imposta le proprietà di sistema relative al proxy con gli eventuali valori presenti nel file.

MIB30 IN REALTIME

Adesso che abbiamo tutti gli ingredienti, possiamo iniziare a implementare l'applicazione concreta. L'idea è quella di creare una dialog che mostri in una tabella le quotazioni delle azioni del Mib 30, specificando il nome dell'azione, l'ultimo prezzo, la variazione rispetto al giorno precedente e l'orario dell'ultimo ordine eseguito. L'applicazione aggiornerà automaticamente i valori ogni cinque minuti o su richiesta dell'utente. Inoltre, per facilitare la visualizzazione, le variazioni positive saranno visualizzate in verde, quelle negative in rosso. Realizzeremo questa applicazione utilizzando le Swing ed in particolare la classe *JTable* per rappresentare le quotazioni delle azioni. La tabella verrà successivamente inserita in una dialog che l'aggiornerà ogni cinque minuti o alla pressione del tasto *"Aggiorna".*

LA STOCKTABLE

Ad esclusione del metodo *getStockQuote,* che restituisce un oggetto di tipo *StockQuote,* tutti gli altri metodi dell'interfaccia *StockConnector,* tra cui *getMib30,* restituiscono un array di oggetti *StockQuote.* La tabella relativa alla nostra applicazione potrebbe quindi essere popolata a partire dagli elementi di un array *StockQuote;* in questo modo, essa potrà essere riutilizzata per visualizzare qualsiasi insieme di azioni. Il modello associato alla *JTable* sarà il seguente:

```
class StockTableModel extends AbstractTableModel {
  private String columns[] =
    { "Azione", "Quotazione", "Variazione %", "Ora"};
  private StockQuote quotes[] = new StockQuote[0];
  public int getColumnCount() {return columns.length;}
  public int getRowCount() { return quotes.length;}
  public Object getValueAt (int row, int column) {
    if (column==0)
      return quotes[row].name;
    ...}
  public String getColumnName (int columnIndex) {
    return columns[columnIndex];}
  public void setQuotes(StockQuote quotes[])
  { this.quotes = quotes; }
  private String getTime(Calendar date) {
    ... }}
```

I dati sono rappresentati dall'array *quotes,* che all'inizio sarà vuoto, ma che potrà essere successivamente reimpostato dal metodo *setQuotes.* La tabella avrà quattro colonne: *"Azione", "Quotazione", "Variazione"* e *"Ora".* Il metodo *getValueAt,* richiamato dal framework per la *JTable,* restituirà il campo opportuno della classe *StockQuote* in base al numero di colonna in input. La data relativa al campo *StockQuote. date* verrà formattata dal metodo privato *getTime* e visualizzata nella tabella nel formatto *hh:mm.* L'istanza di *JTable* che utilizza questo modello verrà inserita in un *JPanel* denominato *StockTable.* L'implementazione la trovate nel CD allegato. Il pannello contiene la *JTable* riferita da *table* che viene creata utilizzando il modello *StockTableModel* implementato in precedenza. La tabella

```
T.I.M.  4.64 -0.53% 22/6/2004 12:50
T.I.M. RNC  4.49 -0.04% 22/6/2004 12:48
TARGETTI SANKEY  3.42 0.0% 22/6/2004 9:20
TAS  16.5 0.01% 22/6/2004 12:13
TC SISTEMA  0.0 0.0% 22/6/2004 0:00
TDIAAW DIR 16.7.02 0.0 0.0% 22/6/2004 0:00
TECNODIFFUSIONE ITA 0.0 0.0% 22/6/2004 0:00
TELECOM IT MEDI RNC 0.0 0.0% 22/6/2004 0:00
TELECOM ITAL MEDIA 0.342 0.0% 22/6/2004 12:49
TELECOM ITALIA 2.585 - 0.07% 22/6/2004 12:50
TELECOM ITALIA RNC 1.855 0.16% 22/6/2004 12:49
TENARIS 2.7 -1.27% 22/6/2004 12:41
TISCALI  3.6 -0.22% 22/6/2004 12:50
TOD'S 27.79 -0.1% 22/6/2004 12:40
TREVI FINANZ INDUST 1.05 1.84% 22/6/2004 12:28
TREVISAN COMETAL 3.53 -0.53% 22/6/2004 11:17
TXT E-SOLUTIONS  19.17 0.0% 22/6/2004 12:25
```

Fig. 1: ***L'output di Connector***

verrà posta sul pannello all'interno di un *JScrollPane*. Il metodo *reload* ha il compito di ripopolare la tabella con i dati passati nell'array in input. Esso richiama il metodo *setQuotes* del modello ed effettua un aggiornamento della tabella. Questo metodo verrà invocato ogni qualvolta le quotazioni verranno prelevate dal provider. Come si può notare, il costruttore di *StockPanel* richiama il metodo *setDefaultRenderer* per la *JTable* passando un'istanza di *ColoredTableCellRenderer*. Questa classe, implementata da noi, ha il compito di visualizzare in verde le variazioni di prezzo positive e in rosso quelle negative. L'implementazione di *ColoredTableCellRenderer* la trovate sul CD.

LA DIALOG MIB30

Il pannello *StockTable* è del tutto generico e funziona a partire da un qualsiasi array di *StockQuote*. Vedremo in questo paragrafo come realizzare una dialog che utilizzando *StockTable* visualizzi in tempo reale le quotazioni delle azioni relative al Mib30. La classe si chiama *Mib30Dialog* e conterrà la *StockTable* ed un bottone *"Aggiorna"* che consentirà all'utente di aggiornarla. Inoltre, la dialog si aggiornerà automaticamente ogni cinque minuti. Il costruttore della classe avrà il compito di leggere le proprietà, impostare l'interfaccia grafica, prelevare le quotazioni da Internet e mostrarle:

Stock Spy

Azione	Quotazione	Variazione %	Ora
ALLEANZA ASS	9.38	0.35	16:33
AUTOSTRADE	16.31	0.67	16:33
BANCA ANTONVENETA	16.89	1.62	16:33
BANCA FIDEURAM	4.66	1.26	16:33
BANCHE POP UNITE	13.59	1.02	16:33
BCA INTESA	3.19	1.59	16:32
BCO POPO VR E NO	14.0	0.64	16:33
BNL	1.925	2.01	16:33
CAPITALIA	2.58	1.93	16:32
EDISON	1.453	-0.61	16:32
ENEL	6.6	-0.81	16:33
ENI	16.73	-1.12	16:33
FIAT	6.9	2.22	16:33
FINMECCANICA	0.64	2.23	16:33
GENERALI ASS	22.27	0.17	16:33
LUXOTTICA GROUP	13.94	0.56	16:33
MEDIASET	9.48	0.35	16:33
MEDIOBANCA	10.1	1.39	16:33
MEDIOLANUM	5.27	1.65	16:33
MONTE PASCHI SIENA	2.66	0.56	16:32
PIRELLI & C.	0.871	2.67	16:33
RAS	14.89	0.48	16:31
SAIPEM	7.75	-0.42	16:32
SANPAOLO IMI	9.98	1.01	16:33
SEAT PAGINE GIALLE	0.35	2.18	16:33
SNAM RETE GAS	3.56	-0.08	16:32
STMICROELECTRONICS	17.83	1.27	16:33
T.I.M.	4.69	0.47	16:33
TELECOM ITALIA	2.585	0.62	16:33
UNICREDITO ITALIANO	4.09	0.81	16:33

Aggiorna

Fig. 2: ***Mib30Dialog in azione***

```
public Mib30Dialog() throws Exception {
  super("Stock Spy");
  properties = loadProperties();
  connector=(StockConnector) Class.forName(properties
      .getProperty("stock.connector.class")).newInstance();
  setSize(500,560);
  stockTable = new StockTable();
  getContentPane().setLayout(new BorderLayout());
  getContentPane().add(stockTable, BorderLayout.CENTER);
  JPanel pnlBottom = new JPanel(new BorderLayout());
  getContentPane().add(pnlBottom, BorderLayout.SOUTH);
  JButton btnUpdate = new JButton("Aggiorna");
  pnlBottom.add(btnUpdate, BorderLayout.EAST);
  showQuote();
 ...
```

Il metodo *showQuote* invocherà il metodo *getMib30* dello *StockConnector* e passerà le quotazione alla *StockTable* utilizzando il metodo reload:

```
private void showQuote() {
  try {
    connector.connect(properties);
    stockTable.reload(connector.getMib30());
  } catch(Exception ex) {
    ex.printStackTrace(); }}
```

Sempre nel costruttore avverrà l'impostazione dell'*ActionListener* per il bottone *"Aggiorna"*:

```
btnUpdate.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    Mib30Dialog.this.showQuote();}
});
```

e la definizione del thread che ogni cinque minuti aggiorna la tabella.

```
Thread t = new Thread(new Runnable() {
  public void run() {
    for(;;) {
      try {
        // Cinque Minuti
        Thread.sleep(5 * 60 * 1000);
      } catch(Exception ex) {ex.printStackTrace();}
      Mib30Dialog.this.showQuote();} } });
t.start();
```

Il metodo main sarà quindi il seguente:

```
public static void main(String a[]) throws Exception {
  Mib30Dialog dialog = new Mib30Dialog();
  dialog.setVisible(true); }
```

Lanciando l'applicazione, mediante la seguente linea di comando:

```
> java Mib30Dialog
```

verrà visualizzata la dialog di Fig. 2, contenente le azioni del Mib30 e le relative quotazioni.

CONCLUSIONI

L'applicazione appena vista è giusto un esempio che mostra le potenzialità del nostro *StockConnector*. Essa funziona solo con le azioni del Mib30, ma con facili modifiche potrete far in modo che funzioni con tutte le azioni del listino. Infatti, è sufficiente invocare *getQuoteAll*, *getByLetter* oppure *getCustom* al posto di getMib30 e la tabella visualizzerà i dati opportuni. Il prossimo mese aggiungeremo un ultimo tassello all'applicazione *Stock Spy*, ovvero la possibilità di memorizzare i dati all'interno di un database relazionale in modo da ottenere lo storico delle quotazioni per eventuali statistiche ed analisi tecnica.

Giuseppe Naccarato

BIBLIOGRAFIA

• **CORE SWING: ADVANCED PROGRAMMING** ***K. Topley*** **(Prentice Hall) 1999**

• **STOCK SPY - LA BORSA ONLINE (PARTE I) ioProgrammo n. 84** ***G. Naccarato,*** **(Edizioni Master)**

Un Web Service per gestire un torneo di FantaCalcio

Web Service e FantaCalcio

parte terza

Migriamo la nostra applicazione di FantaCalcio verso un’architettura a servizi: progettiamo un Web Service per la consultazione del torneo

Siamo ormai giunti alla terza puntata del percorso dedicato allo sviluppo di una soluzione per la gestione di un torneo di *FantaCalcio*. Ricordiamo che nei precedenti abbiamo seguito passo dopo passo tutte le fasi della progettazione di una classica applicazione per la gestione di un torneo: siamo partiti, infatti, definendo un modello di classi di business ed una base dati relazionale ed abbiamo illustrato la loro definizione mediante l’utilizzo del tool *ObjectRelationalBridge* (OJB) e del database *HSqlDb*. Successivamente abbiamo realizzato un’applicazione grafica di gestione del *FantaCalcio* (dedicata soprattutto all’amministratore del torneo) ed abbiamo implementato alcune principali funzionalità del motore interno. Scopo di questa puntata è quello di realizzare, mediante l’architettura dei Web Services, dei servizi che permettano anche ad altri utenti di poter consultare i dati relativi ad un torneo di *FantaCalcio*. Logicamente, questa è un’occasione per esaminare da vicino un reale utilizzo dei Web Services in un contesto già familiare dato che possiamo sfruttare quanto realizzato finora.

QUALCHE CONCETTO DI BASE

In questa sede non è opportuno approfondire il discorso sulle origini e sui concetti fondamentali che sono alla base dei Web Services; pertanto daremo solo qualche definizione generale. Innanzitutto, stabiliamo che per *servizio* s’intende un’applicazione che espone le sue funzionalità ad un’altra applicazione mediante un’interfaccia di programmazione (API). In altri termini non si tratta d’altro che di una risorsa utilizzabile da un’altra applicazione e non da un utente umano. Tale concetto di servizio è insito nel design dell’applicazione cui ci si riferisce con il nome di SOA. Per far sì che un client possa accedere ad un servizio disponibile in Rete, occorre definire necessariamente delle apposite interfacce di programmazione. Si può utilizzare uno dei middleware di comunicazione realizzati finora (RPC, DCOM, CORBA, RMI) ma questi ultimi, a differenza dei Web Services, soffrono di alcuni problemi di scarsa omogeneità, integrazione e flessibilità. Di seguito riportiamo le principali caratteristiche dei Web Services che ci permettono di capire meglio le loro potenzialità:

- un WS è una risorsa del Web, accessibile mediante un protocollo indipendente e neutrale quale HTTP
- un WS espone un’interfaccia (Web API) utilizzabile da qualsiasi altra applicazione
- un WS è registrato in un opportuno Web Registry. Quest’ultimo permette ai client di localizzare i servizi richiesti
- i WS offrono un forte disaccoppiamento tra le applicazioni in quanto la comunicazione avviene mediante lo scambio di messaggi XML.

Parlando di Web Services ci s’imbatte necessariamente in una serie di acronimi che possono generare un po’ di confusione. Il primo di loro è il WSDL *(Web Services Description Language)*, una grammatica XML che permette di definire l’interfaccia del servizio esposto. Un altro è la specifica UDDI *(Universal Discover Description and Integration)* che permette ad un qualsiasi client di localizzare sulla Rete il servizio richiesto in modo da poterlo utilizzare. L’ultimo acronimo è il SOAP *(Simple Object Ac-*

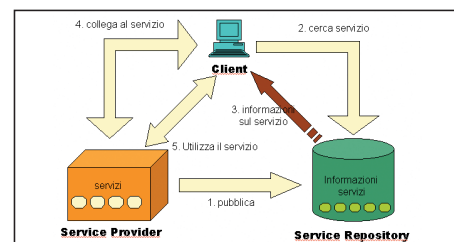



Fig. 1: Un Service Provider ospita un gruppo di Web Services, mentre un Service Repository pubblica delle informazioni che permettono ai client di localizzare il servizio richiesto

cess Protocol): esso, definendo la modalità di accesso ad un Web Service. In particolare si tratta di un protocollo che permette di incapsulare, all’interno della richiesta, un messaggio XML, utilizzando un protocollo di comunicazione standard quale l’HTTP.

JAVA WEB SERVICES E JAX

Mediante Java Web Services, lo sviluppatore Java può costruire delle applicazioni server implementanti le caratteristiche dei Web Services propriamente detti. La Sun mette a disposizione un pacchetto, denominato JWSDP (*Java Web Services Developer Pack*), che consiste principalmente di una serie di API, tra cui le più importanti sono quelle relative al JAX (Java API per XML), in aggiunta ad alcuni tool quali *Jakarta Tomcat Web Container, Ant* ed un *UDDI-based regstry server.* Le API per XML sono distinte in quattro componenti principali:

- **JAXP** e **JAXB**: si occupano del processing e del binding XML.
- **JAXM**: relative al messaging.
- **Registries**: per i registri XML.
- **JAX-RPC**: per le chiamate RPC basate su XML.

GLOSSARIO

RMI

È un modello di architettura distribuita che permette ad un client Java di effettuare chiamate a metodi di oggetti remoti Java come se questi fossero locali. Il modello si basa su due opportune classi generate automaticamente dette *stub* e *skeleton* e che implementano il colloquio tra client e server.

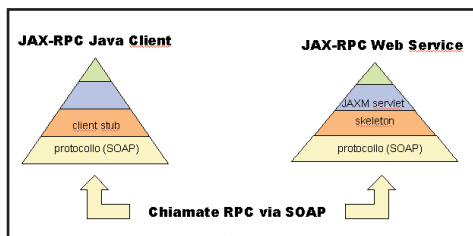



Fig. 2: ***Rispetto al predecessore, JAX-RPC può essere utilizzato per accedere a Web Services non Java.***

L’ultima componente è quella che più ci interessa: altro non è che una specifica per effettuare chiamate a metodi remoti mediante XML e SOAP su protocollo HTTP. In effetti, JAX-RPC è una versione rinnovata del predecessore RMI *(Remote Method Invocation)* che permette ad un client Java di accedere ad un Web Service come se fosse locale. In sintesi, JAX-RPC è in grado di generare WSDL a partire da un’interfaccia Java (e viceversa), nonché ovviamente di produrre le classi di stub e skeleton.

SOA

L’acronimo sta per *Service Oriented Architecture* e definisce un’architettura di base utilizzata nella maggior parte dei sistemi di *middleware* basati su RPC *(Remote Procedure Call)*. La caratteristica più importante del SOA è la separazione dell’interfaccia del servizio dalla sua implementazione.

IL FANTACALCIO SERVICE

A questo punto, procediamo con la progettazione del nostro Web Service mediante il JAX-RPC (nel corso dell’articolo si è utilizzata la versione 1.3 del JWSDP). Scopo del servizio è di permettere ai diversi partecipanti al torneo di consultare diverse informazioni utili quali ad esempio:

- squadre partecipanti e rosa dei calciatori
- calendario della stagione in corso
- prossima giornata da giocare
- classifica del campionato

Il punto di partenza per la creazione di tale servizio è la definizione di un’interfaccia Java (denominata in gergo *service endpoint interface*) che estenda *java.rmi.Remote* e che definisca i metodi esposti.

```
public interface FCalcioIF extends Remote {
  String[] getSquadre() throws
                    RemoteException,BusinessException;
  String getStagioneAttuale() throws
                    RemoteException,BusinessException;
  GiornataBean[] getCalendario(String stagione)
            throws RemoteException,BusinessException;
  GiornataBean getProssimaGiornata(String stagione)
            throws RemoteException,BusinessException;
  CalciatoreBean[] getRosaSquadra(String nome)
            throws RemoteException,BusinessException;
  PuntiSquadraBean[] getClassifica(String stagione)
          throws RemoteException,BusinessException; }
```

Possiamo notare innanzitutto che valgono le medesime regole stabilite per l’RMI: ogni metodo è in grado di scatenare (in aggiunta alla *BusinessException* definita dal nostro modello) un’eccezione di tipo *java.rmi.RemoteException.* Ciò accade proprio perché, come detto, il JAX-RPC è un’estensione di RMI. Nonostante siamo partiti da una classica interfaccia Java, sappiamo che l’interfaccia di un Web Service è definita mediante una particolare grammatica XML denominata WSDL (in realtà si potrebbe anche iniziare, nella creazione di un WS, direttamente da un file WSDL). Dietro le quinte, infatti, il JAX-RPC mappa i tipi di dato Java in definizioni WSDL. Mentre i tipi primitivi (int, long, boolean, ecc.), le classi fondamentali (*String, Integer, Date,* ecc.) e gli array sono completamente supportati, per i tipi più complessi è necessario attenersi ad alcune regole. Senza entrare troppo in dettaglio, diciamo che in tali casi il JAX-RPC permette l’utilizzo di classi personalizzate aderenti allo standard JavaBeans, dotate quindi di metodi *set* e *get* per ogni attributo *private* ed in cui ogni attributo stesso è a sua volta un tipo supportato. Per tale motivo, il tipo di ritorno di alcuni metodi dell’interfaccia *FCalcioIF* è costituito da classi Bean: si tratta di semplici classi JavaBeans, definite nel package *fcalcio. bean,* che contengono un gruppo di informazioni correlate tra loro. Il prossimo passo da compiere è quello di definire un’implementazione per la nostra interfaccia. A tal fine è sufficiente definire una classe *FCalcioImpl* ed in essa implementare i metodi dell’interfaccia *FCalcioIF.* Nel nostro caso, possiamo efficacemente utilizzare quanto realizzato nel corso dei precedenti articoli lavorando sul motore dell’applicazione. Come visibile nel class diagram riportato in Fig. 5, le funzionalità sono già messe a disposizione dall’in-

terfaccia *FCModel*. La classe *FCalcioImpl* deve quindi semplicemente creare un oggetto di *FC Model* e utilizzarne i relativi metodi.

```
public class FCalcioImpl implements FcalcioIF {
  private FCModel model;
  public FCalcioImpl() {
    try {
       model = new FCModelImpl();
    } catch(ODMGException oe)
      { logger.error(oe.getMessage(),oe);} }
```

L'implementazione del metodo *getSquadre*, ad esempio, delega semplicemente il compito al motore stesso.

```
public String[] getSquadre() throws BusinessException
{  logger.info("called getSquadre...");
  return model.getSquadre(); }
```

Viceversa, nel caso di *getCalendario,* l'implementazione deve creare opportune istanze della classe *GiornataBean* a partire dal risultato ottenuto (un oggetto *Campionato*) dalla funzionalità base del motore.

```
public GiornataBean[] getCalendario(String stagione)
                               throws BusinessException {
  logger.info("called getCalendario for stagione "
                                             + stagione);
  Campionato c = model.getCampionato(stagione);
  List giornate = c.getGiornate();
  GiornataBean clb[] = new GiornataBean[giornate.size()];
  for(int k=0;k<giornate.size();k++) {
    Giornata g = (Giornata) giornate.get(k);
    clb[k] = createGiornataBean(g); }
  return clb; }
```

Analogamente avviene per i metodi *getProssimaGiornata* e *getRosaSquadra* che traducono gli oggetti di business (*Giornata* e *Squadra*) ottenuti dal motore in istanze delle classi *GiornataBean* e *CalciatoreBean*. Resta infine il metodo *getClassifica.* Esso deve implementare una logica più complessa consistente nel leggere il calendario (a partire dall'oggetto di business *Campionato*) ed a comporre una classifica di oggetti *PuntiSquadraBean,* rappresentanti la posizione di ogni singola squadra. In questo caso, come possiamo vedere, il metodo utilizza una classe di utilità, denominata *ClassificaComposer.*

Fig. 3: Le classi JavaBeans non sono necessariamente una copia fedele di quelle di business in quanto ne rappresentano solo una particolare vista

```
public PuntiSquadraBean[] getClassifica(String
                      stagione) throws BusinessException {
  logger.info("called getClassifica for stagione " + stagione);
  String squadre[] = model.getSquadre();
  ClassificaComposer composer = new
                           ClassificaComposer(squadre);
  Campionato c = model.getCampionato(stagione);
  List giornate = c.getGiornate();
  composer.run(giornate);
  List cl = composer.getClassifica();
  PuntiSquadraBean[] psb = new
                       PuntiSquadraBean[cl.size()];
  for(int k=0;k<cl.size();k++)
    psb[k] = (PuntiSquadraBean) cl.get(k);
  return psb;}
```

PACKAGING E DEPLOYING

A questo punto, il codice Java per il Web Service è completo. Restano da fare però una serie di operazioni per terminare la fase di packaging ed installazione del nostro servizio. In tale fase entra in gioco *Ant* (nella directory è presente il file *build.xml* necessario ad effettuare il build dell'applicazione) e dovremmo semplicemente creare e/o configurare solo una serie di file. Una volta installato il JWSDP, la prima operazione da fare consiste nel modificare, all'interno del file *build.properties,* le proprietà che indicano il path dell'installazione.

Fig. 4: La definizione di un'interfaccia e di una sua implementazione costituiscono la base di partenza di un Web Service

```
jwsdphome=C:/java/jwsdp-1.3
jwsdpshared=${jwsdphome}/jwsdp-shared
```

In questo modo, *Ant* sarà in grado di effettuare automaticamente il deploy del servizio. Il prossimo passo da compiere è quello di modificare alcuni file di configurazione del motore dell'applicazione, posti nella directory *etc/conf,* e relativi a *Log4j* ed ad OJB. In particolare, nel file *log4j.properties,* contenente la configurazione del logging per il nostro servizio, va specificato il path dei file di log, impostando, ad esempio, la directory d'installazione del JWSDP.

```
log4j.appender.Root=org.apache.log4j.RollingFileAppender
```

OBJECTRELATIONALBRIDGE

È un tool di mapping della Apache Software Foundation per la persistenza di oggetti Java verso un base dati relazionale. Il tool, open source, offre le interfacce ODMG e JDO è disponibile all'indirizzo http://db.apache.org/.

```
log4j.appender.Root.File=C:/java/jwsdp-1.3/logs/root.log
```

Gli altri file di configurazione, presenti sempre nella stessa directory *etc/conf*, sono relativi a OJB e contengono le informazioni necessarie al mapping ed alla connessione al database. Il solo file da modificare è *repository_database.xml* in cui va impostato il path relativo alla cartella contenente il database HSqlDb utilizzato.

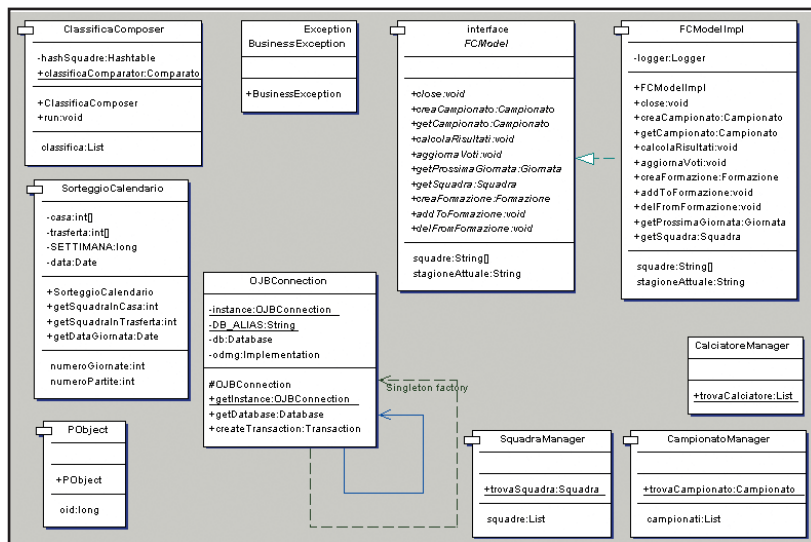



Fig. 5: *Il motore dell’applicazione, sviluppato nei precedenti articoli, è basato sull’interfaccia FCModel e sulla relativa implementazione FCModelImpl*

Possiamo impostare, come indicato sotto, la directory di lavoro di JWSDP

```
subprotocol="hsqldb"
dbalias="C:/java/jwsdp-1.3/work/FantaCalcioDb/
                                    FantaCalcioDb"
```

poiché il processo di build di *Ant* vi copierà un database di prova contenuto nella directory *etc/data*. Prima di procedere con il deploy dell’applicazione esaminiamo brevemente i passi che saranno eseguiti all’interno del build di *Ant*. Poiché un JAX-RPC Web Service in realtà altro non è che un servlet, esso può essere impacchettato ed installato mediante un file WAR. Pertanto, dopo aver compilato i file sorgenti presenti all’interno della directory *src/server*, il build di *Ant* ha anche il compito di creare un normale file WAR al cui interno vanno tra l’altro inserite le librerie ed i file di configurazione (quelli appena visti relativi a Log4j ed ad OJB) necessari al motore dell’applicazione.
Un normale file WAR non è però ancora in grado di definire un Web Service, in quanto non sono presenti la definizione WSDL dell’interfaccia, gli stub, gli skeleton, ecc. Il JAX-RPC mette a disposizione due tool, *wscompile* e *wsdeploy*, che sono in grado di generare queste risorse aggiuntive. Il primo tool, *wscompile*, è in grado di generare i file di mapping (WSDL, stub, skeleton, ecc.) sia per il client che per il server. Tale tool, leggendo in input un file configurazione in XML, è anche in grado di generare un’interfaccia Java a partire da un file. Ad esempio, definendo, per il nostro WS, un file *config.xml* (presente nella directory etc) come:

Fig. 6: *All’interno del JWSDP è presente un’istanza di Tomcat quale Web Container*

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http:
            //java.sun.com/xml/ns/jax-rpc/ri/config">
   <service name="FCalcio" targetNamespace
                        ="http://fcalcio.org/wsdl"
typeNamespace="http://fcalcio.org/types"packageName
                              ="fcalcio.service">
<interface name="fcalcio.service.FCalcioIF"
servantName="fcalcio.service.FCalcioImpl"/>
   </service>
</configuration>
```

comunichiamo a *wscompile* le seguenti informazioni utili a generare i file di mapping necessari al WS:

- il nome del servizio
- il namespace WSDL
- il package contenente le classi del servizio
- il service endpoint interface

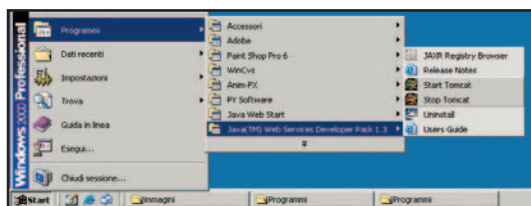
L’altro tool, denominato *wsdeploy*, legge un normale file WAR e, interpretando un file di configurazione (comunemente denominato *jaxrpc-ri.xml* e posto all’interno del WAR di input), genera un altro WAR pronto per il deploy. In realtà, dietro le quinte, esso utilizza il comando wscompile al fine di generare le classi ed il file WSDL da includere poi all’interno del nuovo WAR. Nel nostro caso, il file WAR, creato in precedenza, è quindi processato direttamente dal tool *wsdeploy*, il quale leggerà il nostro file *jaxrpc-ri.xml* (definito nella solita directory etc) ed incluso all’interno del WAR iniziale.

```
<webServices
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
                                    version="1.0"
   targetNamespaceBase="http://fcalcio.org/wsdl"
        typeNamespaceBase="http://fcalcio.org/types"
   urlPatternBase="/fcalcio">
   <endpoint name="FCalcio" displayName=
      "FantaCalcio Service" description="A fantacalcio
         web service" wsdl="/WEB-INF/FCalcio.wsdl"
                 interface="fcalcio.service.FCalcioIF"
       implementation ="fcalcio.service.FCalcioImpl"/>
   <endpointMapping endpointName="FCalcio"
                              urlPattern="/fcalcio"/>
</webServices>
```

SUL WEB

Java Web Services Tutorial
http://java.sun.com/webservices/tutorial.html

JAX-RPC Home
http://java.sun.com/xml/jaxrpc/index.html

Web Services Description Language (WSDL) 1.1 W3C Note
www.w3.org/TR/wsdl

Come si può notare, sono presenti più o meno le stesse informazioni contenute nel *config.xml* definito nel caso si volesse utilizzare il comando *wscompile*. A questo punto possiamo effettuare il build e l’installazione del nostro WS (il file WAR prodotto verrà copiato nella diretory *webapps* del JWSDP) digitando semplicemente

```
ant deploy-war
```

Se non lo abbiamo già fatto, occorre lanciare il server (vedi Fig. 6) e, richiedendo al browser l’indirizzo *http://localhost:8080/jaxrpc-FantaCalcio/fcalcio?WSDL*, potremo visualizzare il file WSDL del servizio.

UN CLIENT PER IL WEB SERVICE

Finalmente è giunto il momento di creare un client in grado di accedere alle funzionalità offerte dal nostro servizio. Il modo più semplice di creare un client per un WS è di legarlo staticamente alla *service endpoint interface* mediante uno stub statico: si tratta di un oggetto proxy che definisce tutti i metodi offerti dalla *service endpoint interface*. Vi sono anche altre modalità di invocazione dinamica, che permettono invece un maggiore disaccoppiamento tra client e service endpoint interface. Nel nostro caso, il client sarà una semplice applicazione di testo che utilizza uno stub generato a tempo di compilazione mediante il tool *wscompile*. Prima di sviluppare il client Java, occorre infatti impostare un file di configurazione (un esempio è il file *config-wsdl.xml* contenuto in etc) in cui è specificata l’URL del file WSDL ed il package del servizio.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
   <wsdl location="http://localhost:8080/
                        jaxrpc-FantaCalcio/fcalcio?WSDL"
  packageName="fcalcio.service" />
</configuration>
```

Il tool *wscompile*, leggendo quanto descritto nel file WSDL ritornato all’URL indicato, crea lo stub e tutto l’occorrente, (quale classi di serializzazione e tipi di dati), per la compilazione del client.
A questo punto possiamo utilizzare, nella classe *FCalcioClient*, lo stub generato facendone il cast all’interfaccia *FCalcioIF* ed invocandone i metodi richiesti.

```
public class FCalcioClient {
  private static Stub createProxy() {
    return (Stub) (new FCalcio_Impl(
                                    ).getFCalcioIFPort());
  }
  public static void main(String[] args) {
    try {
      Stub stub = createProxy();
      stub._setProperty(javax.xml.rpc.Stub
                         .ENDPOINT_ADDRESS_PROPERTY,
  "http://localhost:8080/jaxrpc-FantaCalcio/fcalcio");
  FCalcioIF fcalcio = (FCalcioIF) stub;
  System.out.println("Web Service FantaCalcio");
  String stagione = fcalcio.getStagioneAttuale();
  System.out.println("Stagione attuale: " + stagione);
  ...
```

Digitando il comando

```
ant compile-client
```

otterremo, come descritto, la generazione dello stub e la compilazione del client. Fatto ciò siamo quindi in grado di eseguire l’applicazione client digitando

```
ant run-client
```

CONCLUSIONI

Abbiamo così realizzato, mediante il JAX-RPC, un Web Service del *FantaCalcio* sfruttando quanto messo a disposizione già dal motore interno della nostra applicazione.
Per semplicità l’implementazione del servizio utilizza e gestisce in proprio una connessione al database ma, ovviamente, sarebbe molto più efficiente ottenere tale connessione da un pool gestito dal Web Container (in questo caso Tomcat). Nel corso dell’articolo abbiamo inoltre implementato un client statico facilmente e velocemente realizzabile ma che, come rovescio della medaglia, risulta strettamente legato al servizio. Una possibile direzione di sviluppo potrebbe essere quindi quella di utilizzare una strategia di invocazione dinamica.

David Visicchio

```
C:\WINNT\system32\cmd.exe
D:\Projects\David Jobs\Articoli\Draft\FantaCalcio3\codice>ant run-client
Buildfile: build.xml

run-client:
     [java] Web Service FantaCalcio
     [java] Stagione attuale: 2003-04
     [java] Squadra 1: Forti & Tenaci
     [java] -- Rosa --
     [java] ABBIATI Christian (portiere)
     [java] BALLOTTA Marco (portiere)
     [java] ADANI Daniele (difensore)
     [java] BENARRIVO Antonio (difensore)
     [java] BERTOTTO Valerio (difensore)
     [java] BETTARINI Stefano (difensore)
     [java] BIRINDELLI Alessandro (difensore)
     [java] ALBERTINI Demetrio (centrocampista)
     [java] ALMEYDA Matias (centrocampista)
     [java] AMBROSINI Massimo (centrocampista)
     [java] APPIAH Stephen (centrocampista)
     [java] BACHINI Jonathan (centrocampista)
     [java] BAIOCCO Davide (centrocampista)
     [java] ADRIANO Leite (attaccante)
     [java] AMORUSO Nicola (attaccante)
     [java] BAGGIO Roberto (attaccante)
     [java] BAZZANI Fabio (attaccante)
     [java] Squadra 2: Ventuno
     [java] -- Rosa --
     [java] ANTONIOLI Francesco (portiere)
     [java] BONAIUTI Adriano (portiere)
     [java] BOLIC Drazen (difensore)
     [java] BONERA Daniele (difensore)
     [java] CAFU Marcos (difensore)
     [java] CANDELA Vincent (difensore)
     [java] CANNAVARO Fabio (difensore)
     [java] BARONIO Roberto (centrocampista)
```

Fig. 7: *Nella finestra dei comandi vengono visualizzate tutte le informazioni ottenute interrogando il servizio*

L’AUTORE

David Visicchio è laureato in Ingegneria. Specializzato nella persistenza e nei sistemi middleware di sistemi object-oriented, i suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

VISUAL BASIC

CHIUDERE UN FORM QUANDO PERDE IL FOCUS

(Nel CD: \codice\ CloseForm.zip)

Queste funzioni servono principalmente per gestire un form utilizzandolo come una finestra di pop-up (tipo la lista a discesa da un combobox). Gli eventi *Deactivate* e *LostFocus* non possono essere utilizzati correttamente quando si utilizzano form MDI e modali. Si deve aggiungere al progetto un modulo con le chiamate API necessarie e poi richiamare le funzioni in quattro eventi del form in questione *(Form_Load, Form_Unload, Form_MouseDown, Form_MouseUp)*. Di seguito, per comodità, riportiamo il solo modulo; sul CD-Rom allegato alla rivista e/o sul web (*www.ioprogrammo.it*) potrete reperire l'applicazione completa in formato sorgente *\codice\ CloseForm.zip*

Tip fornito dal sig. R. Roncato

```
Option Explicit

'Per gestire il focus
Private Const GWL_WNDPROC As Long = -4
Private Const WM_NCACTIVATE As Long = &H86
Private Const NO_ACTIVE As Long = 0
```

IL TIP DEL MESE

DA OUTLOOK A VISUAL BASIC

Un tip per prelevare tutti i contatti presenti in Microsoft Outlook 2003 e trasferirli in un'applicazione Visual Basic.

Nel caso in cui si stia sviluppando un'applicazione di tipo manageriale, mantenere aggiornati i contatti tra la nostra applicazione e Microsoft Outlook sarebbe molto positivo!

Tip fornito dal sig. D.Stopponi

```
Option Explicit
'Set references:
'Microsoft Outlook 11.0 Object Library
Dim MyOlApp As Outlook.Application
Private Sub Ottieni_nome(ByRef Nome As String)
    'Ottengo il nome del contatto visto che
    'Outlook mi da "nomecontatto (indirizzo@contatto.it)"
    Dim Pos As Integer
    'Ottengo la posizione della parentesi aperta
    Pos = InStrRev(Nome, "(")
    'Elimino la parte con l'indirizzo
    Nome = Left$(Nome, Pos - 2)
End Sub
Private Sub Form_Load()
    Dim Rubrica As AddressLists
    Dim Record As AddressEntry
    Dim Name As String
    'Apro Microsoft Outlook
    Set MyOlApp = New Outlook.Application
    'Apro la rubrica
    Set Rubrica = MyOlApp.Session.AddressLists
    'Apro il primo contatto
    Set Record = Rubrica.Item(1).AddressEntries.GetFirst
    'Controllo che ci sia almeno un contatto
    While Not (Record Is Nothing)
        Name = Record.Name
        Ottieni_nome Name
        MsgBox Name & vbCrLf & Record.Address
        'Passo al contatto successivo
        Set Record = Rubrica.Item(1).AddressEntries.GetNext
    Wend
    'Contatti terminati
    MsgBox "Indirizzi terminati..."
    'Chiudo il contatto
    Set Record = Nothing
    'Chiudo la rubrica
    Set Rubrica = Nothing
    'Chiudo Microsoft Outlook
    MyOlApp.Quit
    Set MyOlApp = Nothing
End Sub
```

```
Private Declare Function SetWindowLong Lib "user32" Alias
    "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long,
                           ByVal dwNewLong As Long) As Long
Private Declare Function CallWindowProc Lib "user32" Alias
   "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hwnd As
   Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As
                                              Long) As Long
Private lpPrevWndProc As Long
Private frmToClose  As Form
'Per gestire il mouse
Public Declare Function SetCapture Lib "user32" (ByVal hwnd
                                          As Long) As Long
Public Declare Function ReleaseCapture Lib "user32" () As Long
Private Function WindowProcToCloseFormOnLostFocus(ByVal hwnd As
    Long, ByVal iMsg As Long, ByVal wParam As Long, ByVal lParam
                                          As Long) As Long
    WindowProcToCloseFormOnLostFocus = CallWindowProc(
                         lpPrevWndProc, hwnd, iMsg, wParam, lParam)
    If iMsg = WM_NCACTIVATE Then
        If wParam = NO_ACTIVE Then
            Unload frmToClose
            UnhookFormToClose
        End If
    End If
End Function
Public Sub HookFormToClose(frm As Form)
    'Per eliminare eventuali hook precedenti
    If lpPrevWndProc <> 0 Then
        UnhookFormToClose
    End If
    'Hook al form frm
    lpPrevWndProc = SetWindowLong(frm.hwnd, GWL_WNDPROC,
AddressOf WindowProcToCloseFormOnLostFocus)
    Set frmToClose = frm
End Sub
Public Sub UnhookFormToClose()
    If lpPrevWndProc <> 0 Then
        Call SetWindowLong(frmToClose.hwnd, GWL_WNDPROC,
                                                lpPrevWndProc)
        Set frmToClose = Nothing
        lpPrevWndProc = 0
    End If
End Sub
```

JAVA

COMPRIMERE UNA DIRECTORY

Attraverso il package *java.util.zip*, Java fornisce alcuni semplici meccanismi per comprimere e decomprimere file. Il metodo *zipDir* mostra come comprimere ricorsivamente un albero di directory. Il metodo accetta un oggetto *String* come directory da comprimere e un oggetto *ZipOutputStream* che rappresenta il file compresso. Il metodo *zipDir* non include le directory vuote nell'archivio zip creato.

```
try
{
//crea un ZipOutputStream nel quale comprimere i dati
ZipOutputStream zos = new
ZipOutputStream(newFileOutputStream(".\\curDir.zip"));
//partiamo dal presupposto che esista una directory
chiamata inFolder
//chiama il metodo zipDir
zipDir(".\\inFolder", zos);
//chiude lo stream
zos.close();
}
catch(Exception e)
{//cattura le eccezioni}
//codice per il metodo
public void zipDir(String dir2zip, ZipOutputStream zos)
{
try
{
//crea un nuovo oggetto File
zipDir = new File(dir2zip);
//ottiene l'elenco del contenuto delle directory
String[] dirList = zipDir.list();
byte[] readBuffer = new byte[2156];
int bytesIn = 0;
//il loop seguente scansiona ricorsivamente l'albero delle
//directory e comprime i file individuati
for(int i=0; i<dirList.length; i++)
{
File f = new File(zipDir, dirList[i]);
if(f.isDirectory())
{
//se l'oggetto file è una directory chiama nuovamente la funzione
//per aggiungere nell'archivio tutto il contenuto delle directory
//ricorsivamente
String filePath = f.getPath();
zipDir(filePath, zos);
//ripete nuovamente il ciclo
continue;
}
//se arriva qui, significa che l'oggetto File f
//non è una directory,
//quindi crea un FileInputStream invece di f
FileInputStream fis = new FileInputStream(f);
ZipEntry anEntry = new ZipEntry(f.getPath());
zos.putNextEntry(anEntry);
//scrive il contenuto del file in ZipOutputStream
while((bytesIn = fis.read(readBuffer)) != -1)
{
zos.write(readBuffer, 0, bytesIn);
}
//chiude lo stream
fis.close();
}
}
catch(Exception e)
{
//cattura
```

VISUALIZZARE DOCUMENTI PDF DA UN'APPLET

La soluzione proposta utilizza il metodo *showDocument()* dell'interfaccia pubblica *AppletContext* per visualizzare documenti PDF direttamente in una applet Java. Inoltre, necessita di un browser con installato il plugin di Acrobat Reader.

```
package play.pdf;
import java.applet.*;
import java.awt.*;
import java.net.URL;
public class PDFViewer extends Applet {
Font font = new Font("Dialog", Font.BOLD, 24);
String str = "PDF Viewer";
int xPos = 5;
public String getAppletInfo() {
return "PDFViewer\n" +
"\n" +
"File creato con....\n" +
"";
}
public void paint(Graphics g) {
g.setFont(font);
g.setColor(Color.black);
g.drawString(str, xPos, 50);
}
public void start() {
super.start();
try {
URL url = new URL( "http://www.generic.com/documen
to.pdf" );
this.getAppletContext().showDocument( url, "_blank" );
} catch (Exception e) {
System.err.println( "Errore: Impossibile visualizzare il
documento!" );
}
}
```

ASP

FILTRARE UN RECORDSET

È possibile filtrare il risultato di una query su un recordset utilizzando la proprietà *Filter* al posto della clausola *WHERE.*

```
Dim objRS, objFilter ' As ADODB.Recordset
Dim dcnDB ' As ADODB.Connection
Set dcnDB = Server.CreateObject("ADODB.Connection")
dcnDB.Open "Some connection string"
Set objRS = Server.CreateObject("ADODB.Recordset")
objRS.Open "SELECT * FROM tblOrders", dcnDB, adOpenStatic
objRS.Filter = "CustomerName Like %Smith%"
Set objFilter = objRS
```

ACCESS

RICERCA FULL-TEXT IN UN DB ACCESS

Anche se SQL Server è uno dei database più robusti e scalabili per la ricerca full-text, molti sviluppatori continuano ad utilizzare Access quando si tratta di applicazioni che necessitano di database piccoli e poco complessi. È possibile effettuare ricerche full-text all'interno di un database Access semplicemente concatenando i nomi dei campi all'interno delle query SQL. Il codice seguente utilizza una query SQL nella quale combina i campi *FirstName* e *LastName* utilizzando la parola chiave *LIKE,* dopodichè assegna il contenuto della query al metacarattere *an.* Il codice seguente utilizzato all'interno del database *NorthWind* ottiene l'elenco degli impiegati il cui nome contiene i caratteri *"an".*

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<!—#include File="adovbs.inc"—>
<%
Set conn = Server.CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.Recordset")
conn.Open Application("Connection5_ConnectionString")
rs.CursorType = adOpenStatic
rs.LockType = adLockReadOnly
rs.CursorLocation = adUseClient
Set rs.ActiveConnection = conn
rs.Source = "SELECT * FROM Employees WHERE FirstName +
Lastname LIKE '%an%'"
rs.Open
Do While Not rs.EOF
Response.Write rs("FirstName") & " " & rs("LastName") & "<br>"
rs.MoveNext
Loop
rs.Close
conn.Close
Set rs = Nothing
Set conn = Nothing
%>
</BODY>
</HTML>
```

DELPHI

FORMATTARE UN FLOPPY DISK

La procedura proposta, che altro non è che la risposta

all'evento *"OnClick"* di un bottone *(Button1)* messo su di una form chiamata *Form1*, mostra come utilizzare Delphi per formattare un dischetto. Come si può notare, la *ShellExecute* esegue il programma *Rundll32.exe* contenuto, nel nostro caso, nella cartella *C:\Winnt \system32*. Nella Uses principale del programma è necessario dichiarare la unit *ShellApi*.
Il programma è stato testato su Windows 2000.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShellExecute(Application.Handle,Pchar('Open'),
  Pchar('C:\Winnt\system32\Rundll32.exe'),
  Pchar('Shell32.dll,SHFormatDrive'),
  Pchar('C:\Winnt\system32'),SW_SHOWNORMAL);
end;
```

RIEMPIRE DI ZERI UNA VARIABILE DI TIPO STRING CONTENENTE NUMERI

Spesso gli utenti si trovano a digitare codici numerici preceduti da un certo numero di zeri e chiedono al programmatore un metodo per evitare di digitare tutti gli zeri.
La funzione che segue, messa in un programma Delphi, non fa altro che riempire di zeri una variabile di tipo string che contiene un numero. Come si può osservare la funzione ha due parametri: il primo è la stringa da riempire di zeri, il secondo indica il numero di caratteri di cui dovrà essere costituita la stringa restituita dalla funzione.
Per esempio, se il primo parametro contiene il numero 45 e il secondo contiene il valore 8, la stringa restituita dalla funzione sarà 00000045.

```
function RiempiDiZeri(S:String;NumCar:Integer):String;
Var
  SNum:Integer;
  SStr:String;
begin
  Result:=S;
  if S<>'' then
  begin
    SNum:=StrToInt(S);
          SStr:=StringOfChar('0',NumCar)+IntToStr(SNum);
    Result:=Copy(SStr,(Length(SStr)-NumCar)+1,NumCar);
  end;
end;
```

CREARE UN MENU PARTENDO DA UN FILE DI TESTO

Spesso, particolarmente nei programmi gestionali, si ha la necessità di creare dei menu in maniera dinamica. Per esempio, in un software che funziona in rete si potrebbe decidere di far vedere ai singoli utenti solo le scelte nelle quali sono autorizzati ad entrare.
Questo scopo può essere facilmente raggiunto creando un menù diverso per ciascun utente e memorizzando i vari menù su altrettanti file di testo. Le indicazioni riportate di seguito illustrano un possibile modo di conseguire questo risultato.
Salvare in un file di testo chiamato *MENU.TXT* le seguenti informazioni:

```
1,Menù Generale,0
2,Addizione,1
2,Sottrazione,2
2,Moltiplicazione,3
2,Divisione,4
2,-,0
2,Uscita Programma,5
```

Il file *MENU.TXT* deve risiedere nella stessa cartella in cui andremo a salvare il programma Delphi.
Come si può osservare, il file è stato suddiviso in tre colonne: la prima contiene il livello del menù a cui appartiene ciascuna scelta, la seconda contiene la descrizione delle singole scelte e la terza è dedicata al numero della scelta.
Nella penultima scelta è stato messo un trattino nella seconda colonna ad indicare che nel menù dovrà comparire una riga di separazione fra le prime quattro scelte *(Addizione, Sottrazione, Moltiplicazione, Divisione)* e l'ultima scelta *(Uscita Programma)*.
Nella *Uses* principale del programma è necessario dichiarare la *unit Menus*.
Introdurre una sezione *Var*, subito prima di *Implementation*, scrivendo il seguente codice:

```
var
 FMENU0001M: TFMENU0001M;
 MioMenu: TMainMenu;
 MenuGen:TMenuItem;
 SottoScelte:TMenuItem;
 MenuDinamico:TStringList;
```

Nella sezione *Type* del programma dichiarare le tre procedure che seguono:

```
procedure CreaMenu(FileMenu:String);
procedure CreaSubScelte(Nome:String; Numero:Integer);
procedure SceltaClick(Sender: TObject);
```

Scrivere anche il codice relativo alle tre procedure nel modo seguente:

```
procedure TForm1.CreaMenu(FileMenu: String);
Var
 Riga, NomeScelta, Livello:String;
 i, NumScelta, Posizione:Integer;
begin
 // Leggo il Menu dal File di Testo
 MenuDinamico:=TStringList.Create;
 MenuDinamico.LoadFromFile(FileMenu);

 // Creo il Menu
```

```
MioMenu.Free;
MioMenu:=Nil;
MioMenu:= TMainMenu.Create(Self);

// Creo le Scelte
For i:=0 to MenuDinamico.Count-1 do
begin
  Riga:=MenuDinamico.Strings[i];
  Posizione:=Pos(',',Riga);
  Livello:=Copy(Riga, 1,Posizione-1);
  Riga:=Copy(Riga, Posizione+1,Length(Riga)-Posizione);
  Posizione:=Pos(',',Riga);
  NomeScelta:=Copy(Riga,1,Posizione-1);
  NumScelta:=StrToInt(Copy(Riga,Posizione+1,1));

  // Creo le Scelte di Primo Livello
  if Livello = '1' then
  begin
    MenuGen := TMenuItem.Create(Self);
    MenuGen.Caption := NomeScelta;
    MioMenu.Items.Add(MenuGen);
  end;
  // Creo le Scelte di Secondo Livello
  if Livello = '2' then
    CreaSubScelte(NomeScelta,NumScelta);

end;
MenuDinamico.Free;
MenuDinamico:=Nil;

end;

procedure TForm1.CreaSubScelte(Nome:String; Numero:Integer);
begin
 SottoScelte := TMenuItem.Create(Self);
 SottoScelte.Caption := Nome;
 SottoScelte.Tag:=Numero;
 SottoScelte.OnClick := SceltaClick;
 MioMenu.Items[0].Add(SottoScelte);
end;

procedure TForm1.SceltaClick(Sender: TObject);
Var
 I:Integer;
begin

 for I := 0 to MenuGen.Count-1 do
  begin
   if MenuGen.Items[I].Checked then
    MenuGen.Items[I].Checked:=False;
  end;

 TMenuItem(Sender).Checked:=True;

 if TMenuItem(Sender).Tag=1 then
 begin
  // Codice da Eseguire per la Prima Scelta
 end;
if TMenuItem(Sender).Tag=2 then
 begin
  // Codice da Eseguire per la Seconda Scelta
 end;

 if TMenuItem(Sender).Tag=3 then
 begin
  // Codice da Eseguire per la Terza Scelta
 end;

 if TMenuItem(Sender).Tag=4 then
 begin
  // Codice da Eseguire per la Quarta Scelta
 end;

 if TMenuItem(Sender).Tag=5 then
  Form1.Close;
end;
```

Nell'evento *On Create* della Form scrivere il seguente codice:

```
CreaMenu('MENU.TXT');
```

JPEG a rischio di virus, il mito diventa realtà!

Metti il virus nella JPEG

Nel 1994 una e-mail metteva in guardia da un fantomatico virus annidato nelle JPEG. Oggi, nel 2004, un bug dei sistemi Microsoft ha trasformato in realtà ciò che era solo una leggenda metropolitana!

Quante immagini allegate alle e-mail ricevete ogni giorno? Di sicuro tante. E quante volte vi è stato detto che i virus non possono annidarsi nelle immagini? Innumerevoli! Da oggi però le cose cambiano, perché è giunto il momento di temere e diffidare anche delle immagini. A distanza di qualche mese dagli eventi e dalle conseguenze scaturite dal bollettino di sicurezza MS04-028, mi chiedo cosa potranno mai pensare i poveri utenti di Microsoft quando scopriranno di poter essere infettati semplicemente visualizzando un'immagine JPEG! Una innocua immagine può ora trasformarsi in una micidiale arma in mano agli hacker e portare con sé virus, worm, trojan e qualsiasi altra minaccia. Su questa vulnerabilità legata al processore di immagini JPEG di Microsoft (chiamato GDI+) e su tutta la vicenda del bollettino MS04-028 discuteremo ampiamente in queste pagine: analizzeremo uno degli overflow (…l'ennesimo!) che potrebbe causare una mega-infezione di massa simile a quella del worm Sasser, avvenuta qualche tempo fa.

GDIPLUS, UNA LIBRERIA TROPPO CONDIVISA

Leggendo per la prima volta il bollettino MS04-028 sul sito Microsoft, la cosa che balza subito agli occhi è l'incredibile serie di prodotti affetti dalla vulnerabilità: Windows XP (con e senza Service Pack 1), Windows Server 2003, tutta la famiglia di Office (2000, XP e 2003), Project, Visio, Visual Studio, addirittura il .NET Framework… non si finisce più! Ma perché tutto questo? È possibile che una manciata di byte errati nel codice di una DLL possano mettere in ginocchio la serie completa di prodotti di una mega-softwarehouse quale è Microsoft? Paradossalmente, Windows 2000, considerato da Microsoft ormai un antenato ormai da dimenticare, fino al Service Pack 4 non risulta affetto da tale bug, perché casualmente non utilizza e non dispone del componente vulnerabile che ha fatto nascere tutto il problema, ovvero la libreria *GDIPLUS.DLL*.

GDI+ *(Graphics Device Interface)* è il componente di sistema che (assieme a *GDI32.DLL*) fornisce la maggior parte di funzionalità nel processamento di immagini e nel trattamento di oggetti grafici in generale. Tanto per capirci *GDIPLUS.DLL* esporta ben 609 funzioni disponibili a qualsiasi programma che abbia bisogno di manipolare bitmap, effettuare trasformate di matrici, di texture, disegnare linee, poligoni, ellissi e tutto ciò che in generale ha a che fare con la grafica. Pare infatti che questa libreria sia talmente utile da essere sfruttata da una serie di prodotti di terze parti (tipo *ArchiCAD, ACDsee,* ecc.) in maniera intensiva. Il bug scoperto in GDI+ risiede in una funzione dedicata al processamento delle immagini JPEG, in particolare quella usata da EXPLORER per estrarre i commenti testuali racchiusi nei file .JPG mentre si naviga nelle cartelle del disco fisso. L'esistenza di una vulnerabilità in un componente critico (e soprattutto tanto condiviso) come questo rappresenta naturalmente un grave problema per Windows, anche perché – come la storia ci insegna – quando tutto il mondo si ritrova ad usare lo stesso sistema operativo (e la stessa libreria buggata!), automaticamente c'è il rischio di dover fronteggiare infezioni di massa su scala mondiale, causate da worm in grado di sfruttare tale vulnerabilità

UN BUG "VECCHIO" DI QUATTRO ANNI!

La vulnerabilità di GDI+ fa la sua comparsa sulla mailing list *Full-Disclosure* intorno alla metà di Settembre 2004, quasi in concomitanza col bollettino *MS04-028.* L'advisory viene postato dall'esperto di sicurezza Nick DeBaggis, che decide di non includere nessun exploit, fornendo tuttavia tutti i dettagli tecnici sul problema.

SERVICE PACK 2... IMMUNE, MA NON TROPPO!

Windows XP con Service Pack 2 è immune al problema di *GDIPLUS* perché contiene una versione fixata della libreria. Tuttavia molti si chiedono cosa potrebbe succedere ad un sistema WinXP-SP2 dove viene installato - in un secondo momento - Office XP o Visio, che riporrebbero sul sistema una versione difettosa di GDI+.

NOTA

GDI+ SCANNER TOOL

Poiché si è appurato che nel sistema operativo potrebbero essere presenti diverse versioni della stessa libreria *GDIPLUS.DLL* (installate magari a software di terze parti), sia Microsoft, sia l'istituto SANS, hanno rilasciato due tool in grado di rilevare le versioni difettose di GDI+. I tool sono reperibili ai seguenti indirizzi: http://isc.sans.org/gdiscan.php www.microsoft.com/security/bulletins/200409_jpeg.mspx

NON SOLO GDIPLUS.DLL

Il ricercatore Kile Quest ha pubblicato una analisi dettagliata dell'overflow delle JPEG, scoprendo che esistono altre librerie suscettibili al problema *(MSO.DLL* e *VGX.DLL).* Nel suo ambiente di test è riuscito a creare un'immagine JPEG malformata in grado di eseguire codice remoto attraverso la semplice visione con Internet Explorer. Il documento si può reperire su www.unital.com/research/ms_jpeg.pdf.

La prima cosa che desta subito scalpore è l'intestazione dell'advisory che recita più o meno così:

From: Nick D. *To: Full-Disclosure*

Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow

Advisory: September 14, 2004
Reported: October 7, 2003

Si legge "Riportato: Ottobre 7, 2003". Ciò significa che questa vulnerabilità è presente nei nostri sistemi da quasi un anno e che qualcuno ha preferito tenerla nascosta fino ad oggi, piuttosto che divulgarla. A parte questa piccola divagazione, si scopre - indagando più a fondo nell'advisory - che un simile problema di processamento delle JPEG era stato già scoperto nel lontano 2000 nel codice del browser Netscape e che nessuno all'epoca aveva provato a sondare altri prodotti per verificare l'esistenza della stessa vulnerabilità. Questa condivisione di vulnerabilità tra Netscape e Windows nel processamento delle JPEG suona comunque strana: casualità o qualche "copia&incolla" di troppo?

RADIOGRAFIA DI UNA JPEG

Per studiare e capire la vulnerabilità di GDI+, è indispensabile conoscere qualcosa in più su questo formato grafico. Il formato JPEG codifica le immagini in una sequenza ordinata di markers, seguiti dai parametri legati ai markers e da segmenti dati relativi all'immagine, compressi usando particolari trasformate matematiche *(DCT, FFT)*. Un marker è sempre identificato all'interno del file .JPEG dal byte esadecimale *"FF"* ed è seguito sempre da un secondo byte che serve a definire la tipologia di marker. Un elenco dei principali markers presenti nelle JPEG è riportato in *Tabella 1* (l'elenco non è esaustivo). Detto questo, chiunque di voi, aprendo una JPEG con un editor esadecimale, riuscirà a leggere qualcosa in più nel formato e potrà capire come è strutturata. Ogni marker (ad eccezione di *SOI* e *EOI*) è in genere seguito da alcuni parametri e poi dai dati veri e propri. Se consideriamo ad esempio il marker *"FFE0"*, avremo la seguente struttura dati che segue immediatamente dopo il marker :

FF E0 – "JFIF marker"

2 bytes	*L*, lunghezza totale del marker, inclusi i 2 bytes col valore della lunghezza
5 bytes	la string *ASCIIZ "JFIF"* ovvero la sequenza *"4A46494600"*
2 bytes	versione *(01* o *02)*
1 byte	densità (*0=no units; 1=X,Y* sono *dotsXinch, 2=X,Y* sono *dotsXcm)*
2 bytes	*X-density*
2 bytes	*Y-density*
1 byte	*XN-thumbnail (0=no thumbnail)*
2 byte	*YN-thumbnail (0=no thumbnail)*
(3*N) bytes	*N* valori RGB (3 bytes) per i pixel della thumbnail *(N=XN*YN)*

Il marker COM (*FFFE*) è evidenziato nella tabella perché è proprio il punto in cui la libreria GDI+ risulta vulnerabile all'overflow. La definizione di questo marker è molto semplice:

FF FE – "COM marker"

2 bytes	L, lunghezza totale del marker, inclusi i due byte col valore della lunghezza
L-2 bytes	una qualsiasi stringa ASCII, usata come commento.

MARKER	NOME SIMBOLICO	DESCRIZIONE
FF D8	SOI	*Start Of Image* – tutte le immagini .JPG iniziano con questa intestazione
FF E0	JFIF marker	Contiene la sigla ASCII-Z "JFIF" e una serie di informazioni sulla versione del formato usato e sulla densità dei pixel
FF DB	DQT	*Define Quantization Table* – tavola di quantizzazione, contenente in genere 64 bytes
FF C4	DHT	*Define Huffman Table* – tavola con i codici di Huffman usati nella compressione
FF C0 FF C1	SOF	*Start Of Frame*
FF DA	SOS	*Start Of Scan*
FF FE	COM	*Comment Marker* – contiene eventuali commenti testuali (ASCII) inclusi nell'immagine
FF D9	EOI	*End Of Image* – chiusura della JPEG e fine del file

TABELLA 1: Principali marker del formato JPEG

ANALISI DELLA VULNERABILITÀ GDI+

Nell'advisory si legge che ponendo "00 00" o in alternativa *"00 01"* come lunghezza totale *(L)* del marker COM, si può innescare l'overflow della libreria *GDIPLUS.DLL*, quando questa cerca di estrarre e leggere il commento di una JPEG.

L'algoritmo interno della libreria esegue grosso modo questi passi (documentati egregiamente da K. Quest nel suo documento di analisi su GDI+):

1) alloca *(L+2)* byte nell'Heap *H* (dovranno contenere tutta la struttura del marker *COM* più i 2 byte dell'identificatore del marker *"FF FE"*);
2) copia in *DWORD[H]* l'identificatore del tipo di marker (cioè *"FF FE"*);
3) copia in *DWORD[H+2]* i successivi 2 byte, che identificano la lunghezza totale *(L)* del marker;
4) calcola la lunghezza reale del commento come *L-2*, sottraendo dalla lunghezza totale *L*, i due byte che corrispondono al valore della lunghezza;
5) copia i successivi *(L-2)* byte di commento nell'Heap (fatto tramite *memcpy*)

Traducendo in assembly questa sequenza si ottiene il seguente codice, estratto proprio da *GDIPLUS.DLL* (il registro *ESI* contiene proprio il valore *L* nel punto in cui siamo considerando il codice):

```
. . .
lea eax, [esi+2] ; carica in EAX il valore (L+2) – passo (1)
push eax
call sub_588518 ; alloca la memoria richiesta da EAX nell'heap
test eax, eax ; allocazione riuscita?
mov [ebp+lpMem], eax
jz loc_5CC58A
mov cx, word ptr [ebp+arg_4] ; copia il marker "FF
                                          FE" nell'heap
mov [eax], cx
mov cx, word ptr [ebp+arg_0]
lea edx, [esi-2]  ; Len-2 bug!
. . .
mov ecx, edx
mov eax, ecx
shr ecx, 2
rep movsd              ; overflow dell'heap
. . .
```

Il perché usando una lunghezza pari *"00 00"* o *"00 01"* si scatena l'heap overflow adesso dovrebbe essere ora chiaro: il programmatore di GDI+ ha fatto l'assunzione che il valore *L* specificato subito dopo il marker *"FF FE"* sia sempre *L>=2,* proprio perché tale valore, da specifiche, deve includere sempre i 2 byte iniziali contenenti la lunghezza. Un commento con lunghezza zero, avrebbe comunque richiesto un valore *L=2,* di conseguenza al programmatore sarà sembrata una buona idea calcolare al volo, risparmiando qualche istruzione, la lunghezza del commento come *(L-2),* senza effettuare controlli su tale valore (*L* è veramente *>=2* ?) e senza considerare che una JPEG malformata "ad hoc" dagli hacker avrebbe potuto un giorno contenere i valori critici "0000" e "0001". Dal codice mostrato si nota che il valore *[ESI-2]* (contenente *L-2*) viene poi spostato in ECX ed è usato come contatore per pilotare l'istruzione *"REP MOVSD",* che sposta nell'heap – uno ad uno - i restanti bytes del commento JPEG.

Usando una lunghezza pari a 0 (o ad 1), in EDX viene caricato un valore negativo *(0-2 = -2 = 0xFFFE unsigned),* che convertito a 32-bit *(0xFFFFFFFE)* causa uno spostamento nella memoria Heap di un numero spropositato di byte (circa 4 GB) e da qui scaturisce l'overflow. Ulteriori analisi hanno mostrato che l'overflow può essere innescato – con motivi identici a quelli esposti – usando anche i marker *"FF ED", "FF E1", "FF E2".*

EXPLOIT E SHELLCODE DISPONIBILI

Per testare la vulnerabilità basta quindi aprire una qualsiasi JPEG con un editor esadecimale e variare i 2 byte di lunghezza del marker *"FF FE"* impostandoli a "00 00" (o "00 01").

L'overflow si innesca nel momento in cui Windows cerca di estrarre il commento dalla JPEG: ciò avviene in *EXPLORER* quando si sfogliano le cartelle locali e col mouse ci si sofferma su un'immagine JPEG o quando questa viene selezionata. È in questa occasione che viene usata, tramite una serie di chiamate innestate, la funzione difettosa di *GDIPLUS.DLL* col conseguente overflow che termina con un crash di EXPLORER.

A seguito dell'analisi della vulnerabilità sono stati pubblicati, a distanza di pochi giorni l'uno dall'altro, i seguenti exploit, pubblicamente disponibili sul sito di sicurezza *K-Otik* e allegati nella rivista, con alcuni esempi di JPEG malformate (che potrebbero essere segnalate come potenziali virus dai vostri antivirus).

Un breve riassunto della situazione lo trovate riportato nella tabella 2.

EXPLOIT	AUTORE	PAYLOAD	LINGUAGGIO
www.k-otik.com/exploits/09222004.ms04-28.sh.php	Perplexy	Solo Overflow (no shellcode)	Script .sh / Perl
www.k-otik.com/exploits/09222004.ms04-28-cmd.c.php	FoToZ	Lancia una shell *"cmd.exe"*	C++
www.k-otik.com/exploits/09232004.ms04-28-admin.sh.php	Elia Florio	Aggiunta di un utente "X" nel gruppo Administrators	Script .sh / Perl
www.k-otik.com/exploits/09252004.JpegOfDeath.c.php	John Bissell	BindShell, ReverseShell	C++
JPEG Downloader 1.0 (sorgente non divulgato)	ATmaCA (?)	Scarica un file .EXE da un URL Internet e lo esegue	?

TABELLA 2: Exploit correntemente disponibili per la vulnerabilità MS04-028

CONCLUSIONI

Fuggire dall'heap e prendere il controllo dell'esecuzione non è comunque semplice, come nel caso degli stack overflow. Tuttavia, come si è visto nei diversi test eseguiti, l'overflow finisce col controllare il registro *EDX,* che va a scrivere alcuni byte al posto giusto, variando l'indirizzo di ritorno e di conseguenza alterando il flusso dl'esecuzione di *GDIPLUS.* I diversi exploit creati prevedono il salto, tramite l'inserimento del valore di ritorno giusto in *EDX,* ad una locazione dell'heap precisa dove sono caricati i bytes dell'immagine JPEG. Se nell'immagine viene quindi inserito uno shellcode ben congegnato al posto giusto (preceduto da un tappeto di NOP), è possibile farlo eseguire. La lunghezza dello shellcode è stata stimata fino ad un massimo di 2500 byte, che sono tantissimi! L'unica restrizione è che lo shellcode non può contenere la sequenza *"FF D9"* e altri caratteri speciali (*NUL, FF*) che potrebbero essere confusi come campi dell'header JPEG.

Al momento esistono in circolazione diversi tool in grado di trarre vantaggio da questa vulnerabilità, ma non è escluso che in queste ore non faccia la comparsa su Internet il primo e temutissimo JPEG-Worm della storia!

Elia Florio

SUL WEB

K-Otik - Bollettino con exploit e pericoli derivanti dalle JPEG
www.k-otik.com/news/09232004.MS04-028Threat.php

Bollettino ufficiale MS04-028 sul componente GDIPLUS.DLL, con relativa patch
www.microsoft.com/technet/security/bulletin/ms04-028.mspx

Advisory originale del bug sulle JPEG per Netscape (anno 2000)
www.openwall.com/advisories/OW-002-netscape-jpeg.txt

Analisi del bug effettuata da Nick DeBaggis
www.securiteam.com/windowsntfocus/5QP0C1FE0Y.html

Informazioni sul formato JPEG e sui marker
www.obrador.com/essentialjpeg/headerinfo.htm

www.funducode.com/freec/Fileformats/format3/format3b.htm

Un semplice circuito per pilotare il mondo!

Radiocomando per PC

In un mondo dove la connettività diviene sempre più "senza fili" realizziamo un radiocomando per PC, dotato di una portata di oltre 500 metri: con componenti a basso costo e alta affidabilità

Sistemi di sicurezza, porte e cancelli, sistemi di irrigazione e di automazione: sono solo alcune delle possbilità offerte da sistema di controllo proposto questo mese. In questa sede vogliamo realizzare un sistema radiocomandato *ON-OFF* ad un canale, comandato da un computer per mezzo della porta seriale: desideriamo comandare l'accensione di un LED a distanza, che potrà essere sostituito facilmente, attraverso un apposito stadio di potenza, con qualunque attuatore elettromeccanico, come ad esempio servomeccanismi a motore, elettrovalvole e sistemi di illuminazione. Tutte le informazioni necessarie alla realizzazione del sistema saranno contenute in questo articolo, partendo dal circuito elettronico, per giungere al software di controllo: sarò ovviamente a disposizione dei docenti e degli studenti che vogliano approfondire l'argomento sul forum di ioProgrammo, oppure attraverso l'indirizzo di posta elettronica *luca.spuntoni@ioprogrammo.it*.

Fig. 1: I comandi della applicazione di gestione del radiocomando sono semplici ed immediati

L'antenna in questione è detta marconiana (in ricordo di Guglielmo Marconi. Poiché il nostro sistema a radiocomando funziona ad una frequenza di 433.92 Mhz, la lunghezza della nostra antenna sarà pari ad un quarto della lunghezza d'onda, ovvero $L=C/(4*f) =3 *10^8/ (4*4.33928)= 0,173\ m=17,3\ cm$, dove C è la velocità della luce e f la frequenza di lavoro. Con questa antenna, semplice e compatta, ed il circuito proposto in queste pagine, è stata raggiunta una distanza di trasmissione di oltre cinquecento metri senza ostacoli (oltre settanta metri con edifici). Modificando la configurazione dell'antenna ed inserendo un circuito amplificatore, è possibile aumentare considerevolmente la distanza di trasmissione.

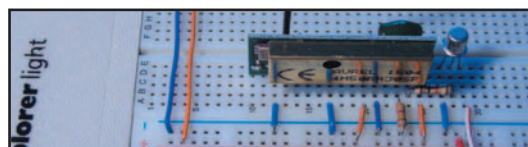



Fig. 2: Un particolare del progetto montato

IL PROGETTO

Il principio di un radiocomando si basa su un trasmettitore a radiofrequenza e un ricevitore, entrambi sintonizzati su una determinata frequenza e dotati di un sistema di trasmissione delle informazioni comune, siano queste analogiche o digitali. Nella nostra applicazione utilizzeremo due moduli integrati a basso costo e alte prestazioni, che permettono di modulare una portante a radiofrequenza, con modalità *ON-OFF*, consentendo la trasmissione di segnali digitali. Come antenna, utilizzeremo un semplice cavo elettrico rigido, di lunghezza opportuna: più precisamente lungo un quarto della lunghezza d'onda della frequenza della portante.

IL TRASMETTITORE

Il trasmettitore del nostro radiocomando è basato sul modulo *TX-SAW-433/s-Z*, della Aurel (*www.aurel.it*) ed è destinato ad un impiego dove sia necessaria la modulazione *ON-OFF* di una portante per la trasmissione di segnali digitali. Il trasformatore è conforme alle direttive europee sull'argomento. Il metodo di controllo del sistema viene ovviamente realizzato in modo tale che possa sfruttare le potenzialità di un Personal Computer attraverso un collegamento tramite la porta seriale. Il circuito elettrico del trasmettitore non necessita di alcuna alimentazione esterna, dal momento che la piccola quantità di potenza necessaria viene prelevata dalla porta seriale del PC. Analizzando infatti lo schema elettrico del trasmettitore, possiamo notare che la linea DTR *(Data Transfer Ready)* della seriale è collegata allo stabilizzatore di tensione *IC1*, che ha il compito di stabilizzare la tensione al valore di 5 Volts, necessario all'alimentazione del modulo trasmettitore. Il diodo *D1* provvede a "bloccare" la tensione negativa

Fig. 3: *Il trasmettitore non necessita di alcuna alimentazione esterna: l'unica connessione necessaria è quella relativa alla porta seriale*

presente sulla linea DTR, quando questa si trova allo stato logico "*LOW*", dal momento che questo valore di tensione può raggiungere anche –25 Volts su alcuni PC. I condensatori *C1,C2* e *C3* provvedono a filtrare ed a livellare la tensione stabilizzata da *IC1* prima che raggiunga il modulo trasmettitore. Il deviatore

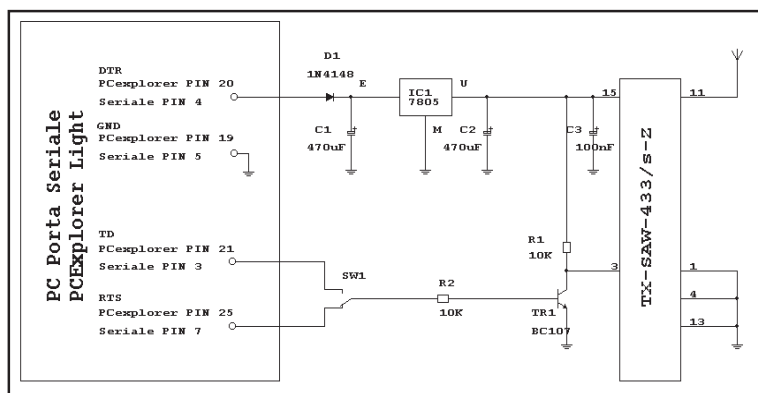



Fig. 4: *Radiocomando: Il trasmettitore ha una portata effettiva di oltre cinquecento metri senza ostacoli ed oltre settanta tra gli edifici*

SW1 permette di selezionare, come linea di trasmissione, *TD (Transmit Data)* oppure *RTS (Request To Send)*. Questa opzione consente di implementare diverse strategie di progettazione, consentendo l'utilizzo della *UART* contenuta nella porta seriale e della sua potenza operativa nel caso si desideri effettuare la trasmissione di dati, oppure di un sistema asincrono qualora si voglia sviluppare un proprio sistema di controllo. A questo scopo il gruppo di componenti *R1, TR1, R2* permette di convertire i segnali provenienti dalla porta seriale a livelli *RS232* (fino a +-25V) in livelli logici *TTL* (0, +5V), oltre che fungere da *NOT* logico. In definitiva, con pochi componenti elettronici, siamo in grado di realizzare un circuito trasmettitore stabile, semplice ed affidabile, capace di essere impiegato in innumerevoli applicazioni di

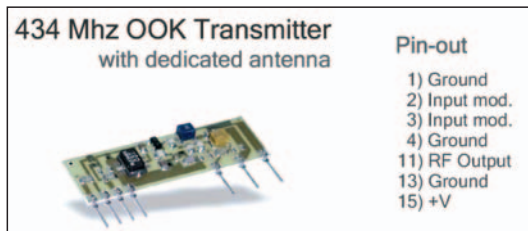



Fig. 5: *La piedinatura del modulo trasmettitore è visibile in figura (cortesia Aurel S.p.A.)*

comando a distanza. Il circuito stampato e la disposizione dei componenti sulla scheda vengono pubblicati in queste pagine, inoltre per facilitarne la realizzazione, sul CD sono disponibili gli schemi dei circuiti in formato bitmap.

IL RICEVITORE

Il ricevitore del radiocomando è stato realizzato utilizzando il modulo *4M50RR30SF* sempre della Aurel assemblando i pochi componenti elettronici per mezzo di PCExplorer light. Lo schema elettrico è davvero molto semplice: il modulo ricevitore viene alimentato dalla apparecchiatura PCExplorer light, dotata di alimentatori stabilizzati interni ed il segnale ricevuto viene utilizzato per pilotare un LED, contenuto all'interno della stessa apparecchiatura attraverso il transistor *TR1* e la relativa resistenza di base *R1*. Il condensatore di disaccoppiamento *C1*, viene posto in vicinanza del modulo ricevitore. Il circuito ricevitore, risulta di facile realizzazione, stabile ed affidabile e verrà ampliato nel prossimo numero al fine di realizzare un sistema completo di trasmissione di dati digitali.

IL RADIOCOMANDO

Abbiamo visto in queste pagine che il nostro radiocomando è composto da un trasmettitore e da un ricevitore: il lettore ha a disposizione tutti gli schemi elettrici e tutto il software necessario alla realizzazione del progetto. Per facilitarne la realizzazione è possibile assemblare i componenti senza saldature per mezzo della apparecchiatura PC Explorer light, oppure utilizzando i soli componenti elettronici elencati a lato di queste pagine, per mezzo del loro montaggio su una comune basetta millefori, utilizzando un saldatore a stagno. L'assemblaggio dei componenti viene facilitato osservando le immagini riportate in queste pagine, disponibili anche nel file allegato alla rivista. Il lettore potrà reperire i componenti elettronici che utilizzeremo per la costruzione del radiocomando in qualunque negozio di com-

Fig. 6: *Ricevitore: lo schema elettrico è molto semplice, garanzia di stabilità e affidabilità*

COMPONENTI ELETTRONICI

TRASMETTITORE

Qnt.	Componente
1	Modulo TX-SAW -433/s-Z
1	7805
1	Transistor BC 107
1	Diodo 1N4148
2	Resistenza 10Kohm 1/4W
2	Condensatori 470 uF 15V
1	Condensatore 100 nF
1	Deviatore a levetta

RICEVITORE

Qnt.	Componente
1	PCExplorer light
1	Modulo 4M50RR30SF
1	Transistor BC 107
1	Resistenza 10Kohm 1/4W
1	Condensatore 100 nF

Per la realizzazione è possibile utilizzare PCExplorer light, l'apposito kit di montaggio, oppure una piastra millefori per montaggi sperimentali.

ponenti elettronici, oppure per corrispondenza presso la Elisys s.r.l. (*www.pcexplorer.it*); i moduli a radiofrequenza sono disponibili anche presso la Aurel S.P.A. Si consiglia di inserire prima i componenti a profilo più basso, iniziando con il circuito integrato e con le resistenze, per poi collegare i condensatori ed il transistor. Le connessioni possono essere realizzate per mezzo di spezzoni di filo elettrico rigido, seguendo lo schema elettrico e le fotografie poste a corredo dell'articolo. Sul lato sinistro dello schema si possono notare le connessioni alle linee relative alla porta seriale e di alimentazione dell'apparecchiatura PC Explorer light.

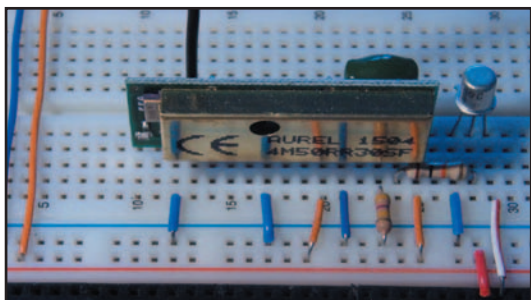



Fig. 8: **Nell'immagine sono visibili le connessioni del ricevitore del radiocomando, realizzato con PCExplorer light**

IL SOFTWARE

Il software di gestione del radiocomando consente di abilitare l'alimentazione del trasmettitore e di modulare la trasmissione di un segnale ad onda rettangolare, che permette l'accensione di un diodo LED, anche a diverse centinaia di metri di distanza: questo segnale digitale può essere facilmente utilizzato per azionare svariati tipi di servomeccanismi, con l'unico limite della fantasia dell'utilizzatore. Di seguito si riporta la classe principale del programma, semplificata in queste pagine, della quale il lettore potrà trovare una versione completa nei file del codice sorgente che il programma di installazione provvederà a posizionare nella cartella di installazione della applicazione. Sono stati utilizzati due componenti sviluppati con Delphi (*SpuntoLedComponent* e *SpuntoHyperLabel*) che vengono distribuiti nel file allegato al CD della rivista in forma compilata. Chi vuole, può comunque indagarne il funzionamento: nel CD sono disponibili anche in formato sorgente. Questi due componenti permettono rispettivamente la gestione dei componenti che operano la simulazione sulla form di un diodo LED e la gestione di collegamenti ipertestuali.

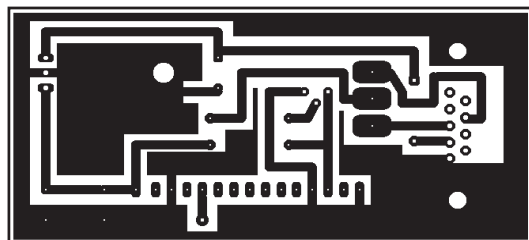



Fig. 9: **Lo schema del circuito stampato per la realizzazione del trasmettitore del radiocomando**

```
Uses
  Windows, Messages, SysUtils, Variants, Classes,
                              Graphics, Controls, Forms,
  Dialogs, ExtCtrls, SpuntoLedComponent, jpeg,
                 Buttons, StdCtrls, SpuntoHyperLabel;
 TSpuntoRadicomandoOnOffForm = class(TForm)
   SpuntoHyperLabel1: TSpuntoHyperLabel;
   SpuntoHyperLabel2: TSpuntoHyperLabel;
   SpuntoHyperLabel3: TSpuntoHyperLabel;
   procedure POWERONbuttonClick(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure WritePort(PortAddress, PortData:word);
   // Write PortData over PortAddress if Port Writing is
                                               enabled
   function ReadPort(PortAddress: word): word;
   procedure RadioButtonCOM1Click(Sender: TObject);
   procedure RadioButtonCOM2Click(Sender: TObject);
   procedure ReadAllPorts;
   procedure Timer1Timer(Sender: TObject);
   //Reads all COM Ports related to selected serial port
   Procedure ExtractPortArray(PortByte:Word; Var
                                  PortArray:TPortArray);
   Procedure EncodePortArray(Var PortByte:Word;
                                  PortArray:TPortArray);
   procedure DTRSpeedButtonClick(Sender: TObject);
   procedure RTSSpeedButtonClick(Sender: TObject);
   procedure SquareWaveScrollBarChange(Sender: TObject);
   procedure TXSquareWaveSpeedButtonClick(Sender:
                                              TObject);
   procedure SquareWaveTimerTimer(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
   CombaseAddress:Word;
   MCRAddress,LCRAddress,MSRAddress:Word;
   MCR,LCR,MSR:TPortArray;
   EnablePortWriting: Boolean;
   RTS,CTS,DSR,CD,DTR,RI:Boolean;
 end;
```

Alla creazione della finestra principale viene eseguita la procedura *FormCreate*, che imposta i valori di default della porta seriale (*COM1*), le impostazioni dei timer e dei pulsanti dell'applicazione.

```
procedure TSpunto RadicomandoOnOffForm.
                            FormCreate(Sender: TObject);
begin
// Port Setup ( COM 1 )
  EnablePortWriting:= True;
  CombaseAddress:=$3f8;
  MCRAddress:=CombaseAddress+4;
  LCRAddress:=CombaseAddress+3;
  MSRAddress:=CombaseAddress+6;
// Timer Setup
  Timer1.Enabled:=False;
  Timer1.Interval:=10;
 // Buttons setup
 ReadAllPorts;
```

CODICE ALLEGATO ALLA RIVISTA

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD e/o sul Web www.ioprogrammo.it all'interno del file di installazione *RadiocomandoPC.exe*.

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata al prezzo di € 213,60 nella versione light, € 99 nella versione basic e € 69 in kit (IVA inclusa) sul web all'indirizzo www.pcexplorer.it oppure inviando una e-mail all'indirizzo pcexplorer@elisys.it, o telefonicamente al numero 0823/468565 o via Fax al: 0823/4954 83

```
 If DTR then DTRSpeedButton.Down:=True else
                       DTRSpeedButton.Down:=False;
 If RTS then RTSSpeedButton.Down:=True else
                       DTRSpeedButton.Down:=False;
 // Waveform Timer Interval
 SquareWaveTimer.Interval:=SquareWaveScrollBar.Position;
 //Transmission Enable at startup or not?
 SquareWaveTimer.Enabled:=
                    TXSquareWaveSpeedButton.Down;
end;
```

La classe *TspuntoRadicomandoOnOffForm* contiene al suo interno due componenti non visuali timer, *SquareWaveTimerTimer* e *Timer1Timer* che svolgono due compiti fondamentali, provvedendo rispettivamente a generare la forma d'onda rettangolare che viene successivamente trasmessa dal trasmettitore ed operare l'aggiornamento dello stato di tutti i componenti visuali della form.

```
procedure TSpuntoRadicomandoOnOffForm.
            SquareWaveTimerTimer( Sender: TObject);
VAR
MCRWord:Word;
begin
ReadAllPorts;
MCR[1]:=Not(RTS);
if TXSquareWaveSpeedButton.down=False then
                                   MCR[1]:=False;
//Codifico MCR in Word e poi lo invio alla porta
EncodePortArray(MCRWord,MCR);
WritePort(MCRAddress,MCRWord);
end;
```

Come è già stato detto in precedenza la trasmissione del segnale avviene variando lo stato logico della linea *RTS*, questa operazione avviene leggendo innanzi tutto lo stato della porta seriale, operando il *NOT* logico della linea *RTS* e riscrivendo il registro MCR (*Modem Control Register*) della porta. L'intervallo di esecuzione degli oggetti timer può essere agevolmente variato modificando la proprietà *Interval* dell'oggetto in questione.

```
procedure TSpuntoRadicomandoOnOffForm.
                    Timer1Timer(Sender: TObject);
Var
CurrentY:Integer;
begin
ReadAllPorts;
If RTS then LedRTS.LedOn else LedRTS.LedOff;
If CTS then LedCTS.LedOn else LedCTS.LedOff;
If DSR then LedDSR.LedOn else LedDSR.LedOff;
If CD  then LedCD.LedOn else  LedCD.LedOff;
If DTR then LedDTR.LedOn else LedDTR.LedOff;
If RI  then LedRI.LedOn else  LedRI.LedOff;
//RF OUT LED
If RTS then RFOUTSpuntoLed.LedOn else
                         RFOUTSpuntoLed.LedOff;
end;
```

L'aggiornamento dello stato logico dei componenti visuali avviene in modo abbastanza ovvio impostando lo stato dei *LED* (*ON* oppure *OFF*) a seconda del livello logico delle linee corrispondenti. Alla pressione del pulsante *Power ON/OFF* viene eseguita la procedura che segue che provvede ad impostare la linea *DTR* a livello logico *HIGH* o *LOW* a seconda che il pulsante sia premuto o meno, alimentando o spegnendo in questo modo il circuito trasmettitore.

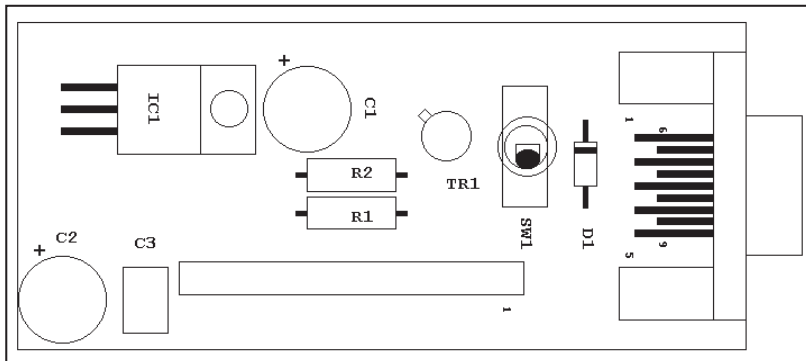
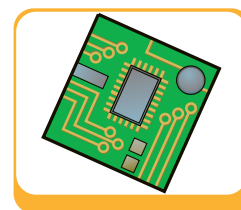
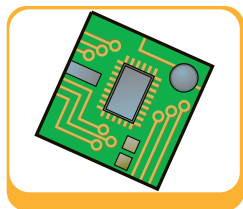



***Fig. 10:** In figura è visibile la distribuzione dei componenti sul circuito stampato del trasmettitore*

```
procedure TSpuntoRadicomandoOnOffForm.
            POWERONbuttonClick(Sender: TObject);
VAR
MCRWord:Word;
begin
SquareWaveTimer.Enabled:=(POWERONbutton.Down
            and TXSquareWaveSpeedButton.Down);
Timer1.Enabled:=POWERONbutton.Down;
ReadAllPorts;
If DTR then DTRSpeedButton.Down:=True else
                    DTRSpeedButton.Down:=False;
If RTS then RTSSpeedButton.Down:=True else
                    RTSSpeedButton.Down:=False;
ReadAllPorts;
MCR[0]:=POWERONbutton.Down; //Power ON/OFF DTR line
if TXSquareWaveSpeedButton.down=False then
                    MCR[1]:=False;    // RTS Line
//Codifico MCR in Word e poi lo invio alla porta
EncodePortArray(MCRWord,MCR);
WritePort(MCRAddress,MCRWord);
end;
```

Il programma permette di modificare manualmente lo stato logico delle linee *DTR* ed *RTS* della porta seriale, per mezzo delle due procedure che seguono, al momento della pressione del rispettivo pulsante.

```
procedure TSpuntoRadicomandoOnOffForm.
            DTRSpeedButtonClick(Sender: TObject);
VAR
MCRWord:Word;
```

PRECAUZIONI

Prima di collegare il circuito al PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto.

```
begin
ReadAllPorts;
MCR[0]:=DTRSpeedButton.Down;
//Codifico MCR in Word e poi lo invio alla porta
EncodePortArray(MCRWord,MCR);
WritePort(MCRAddress,MCRWord);
end;
```

La procedura che segue, eseguita non appena si preme il pulsante *RTS*, ha la caratteristica di consentire il controllo manuale dello stato logico del segnale trasmesso, agendo direttamente sulla linea fisica corrispondente della porta seriale.

```
procedure TSpuntoRadicomandoOnOffForm.
                    RTSSpeedButtonClick(Sender: TObject);
VAR
MCRWord:Word;
begin
ReadAllPorts;
MCR[1]:=RTSSpeedButton.Down;
//Codifico MCR in Word e poi lo invio alla porta
EncodePortArray(MCRWord,MCR);
WritePort(MCRAddress,MCRWord);
end;
```

È possibile variare la frequenza di modulazione della portante, variando la scrollbar relativa: facendo ciò la procedura che segue modifica l'intervallo temporale di azionamento del timer *SquareWaveTimer*, modificando di conseguenza la forma d'onda trasmessa.

```
procedure TSpuntoRadicomandoOnOffForm.
          SquareWaveScrollBarChange(Sender: TObject);
begin
SquareWaveTimer.Interval:=SquareWaveScrollBar.Position;
end;
```

ESECUZIONE SU WINDOWS

L'installazione e l'esecuzione del programma non creano difficoltà, il software viene fornito anche nella versione completamente compilata e collaudata. Il file eseguibile di installazione provvede automaticamente a predisporre la creazione di un'unica cartella configurabile dall'utente, all'interno della quale vengono copiati tutti i file necessari all'utilizzazione del software.
Al termine dell'installazione saranno disponibili i collegamenti alle versioni del programma per Win 9.X /Me, chiamato "*Radiocomando PC*", nonché alla versione per Win XP/NT/ 2000, denominato "*Radiocomando PC XP NT*", oltre che ai collegamenti agli schemi elettrici della realizzazione. Durante l'esecuzione del programma, tutti i comandi sono disponibili in un'unica form: partendo dall'alto notiamo i radio buttons deputati alla selezione della porta seriale alla quale è collegata il circuito elettronico, con i relativi indirizzi di default per *COM1* e *COM2*. Nel caso in cui i parametri non coincidano con quelli del computer che si sta utilizzando, il lettore non avrà difficoltà di modificarne il valore nelle procedure *RadioButtonCOM1click* e *RadioButtonCOM2 click*. I due pulsanti presenti sul lato sinistro della form provvedono rispettivamente ad alimentare il circuito trasmettitore (*Master Power ON /OFF*) e ad azionare la trasmissione del segnale digitale (*Canale 1 ON/OFF*). La frequenza di trasmissione del segnale può essere variata modificando la posizione della scrollbar posta in basso alla finestra. L'utilizzo del programma su sistemi dotati di Win 2000, XP oppure NT viene reso possibile dal programma di installazione per mezzo delle opportune predisposizioni: occorre tenere presente che per potere operare l'installazione su queste macchine occorre disporre dei relativi privilegi di sistema.
Poiché il software accede direttamente all'hardware della macchina, viene installato il driver "*PortTalk - A Windows NT/2000/XP I/O Port Device Driver Version 2.2*" che può essere anche scaricato dal sito: '*www.beyondlogic.org /porttalk/ porttalk.htm*'. Il file '*Porttalk22.zip*' contiene tutte le istruzioni necessarie all'utilizzo corretto del driver, nonché una notevole mole di informazioni relative all'accesso delle porte hardware del PC: viene fornito inoltre il codice sorgente completo delle applicazioni.
Per quanto riguarda l'utilizzo del programma proposto in questa sede sotto Win 2000/NT/XP vengono estratti i file contenenti il driver *Porttalk* nella directory del programma da utilizzare e viene copiato il file *porttalk .sys* nella directory di sistema. Per consentire l'esecuzione del programma viene chiamata l'applicazione *Radiocom.exe* per mezzo del comando inserito nel file batch *Radiocomando-XP.bat: Allowio Radiocom.exe /a* che consente l'accesso da parte dell'eseguibile *Radiocom.exe* a tutte le porte del PC.

CONCLUSIONI

Nel prossimo appuntamento introdurremo le prime applicazioni reali di questo progetto. Un doveroso e sentito ringraziamento è dovuto alla AUREL S.p.A. per la cortese autorizzazione alla pubblicazione della documentazione relativa ai moduli *TX-SAW 433/s-Z 434* e *RX-4M50RR30SF*. Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per ogni eventuale conseguenza.

Luca Spuntoni

BIBLIOGRAFIA

- TX-SAW 433/S-Z 434 MHZ OOK TRANSMITTER (Aurel) 2001
- HIGH PERFORMANCE & LOW COST RECEIVER RX-4M50RR30SF (Aurel) 2001
- PRACTICAL RF CIRCUIT DESIGN FOR MODERN WIRELESS SYSTEM, Vol I e II, *Besser-Gilmore*, (Artech House Books) 2003
- RF DESIGN GUIDE *P. Vizmuller* (Artech House Books) 1995

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo: luca.spuntoni@ioprogrammo.it

Siti WEB facilmente aggiornabili e visibili su cellulari i-mode

Creare siti per i-mode

parte seconda

In questa seconda parte porteremo a termine la progettazione di un sito aggionabile e gestibile in remoto. Realizzeremo pagine i-mode compatibili per rendere disponibili i servizi del sito anche via cellulare

Nello scorso numero abbiamo analizzato le problematiche relative alla gestione di un sito da remoto: abbiamo realizzato pagine ASP accessibili solo all'amministratore e predisposto l'interfaccia utente-studente. Abbiamo anche visto come utilizzare componenti software per l'invio di e-mail da pagine ASP. In questa seconda parte completeremo la costruzione del sito creando pagine per l'interfaccia utente-docente e realizzeremo una sezione per offrire agli utenti-studenti servizi fruibili anche da cellulari dotati di tecnologia i-mode.

INSERIMENTO ARTICOLI

Ogni docente può inviare al server i suoi articoli. Questi saranno inclusi nel database e consultabili dagli studenti. Per inserire i dati, il docente utilizza il form *InserimentoArticoli.asp:* deve specificare anche il nome del file di testo (deve contenere solo testo, senza marcatori HTML) che contiene l'articolo e di cui sarà eseguito l'UpLoad. Ci viene in aiuto di nuovo il componente *Persits:* diamo uno sguardo al file *Esegui_Inserimento_Articoli.asp.*

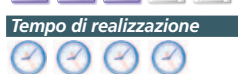
```
application.Lock
set Upload = Server.CreateObject("Persits.Upload")
   p_count = Upload.Save("c:\inetpub\
                        wwwroot\ifp\docenti\articoli\temp")
   p_codiceCorso=Cstr(Upload.form("p_codiceCorso"))
   p_file= Cstr(Upload.form("p_file"))
   p_username= Cstr(Upload.form("p_username"))
   p_password= Cstr(Upload.form("p_password"))
   p_TitoloArticolo= Cstr(Upload.form("p_TitoloArticolo"))
   p_subfileName=Upload.Files(1).Path
   p_subfileName1=replace(p_subfileName,"c:
         \inetpub\wwwroot\ifp\docenti\articoli\temp\", "")
   set Upload=Nothing
```

Si crea una istanza dell'oggetto *Persits.Upload* e si carica il file con il metodo *Upload.Save*, specificando la cartella in cui salvarlo. Prima di eseguire l'upload, si blocca l'applicazione per evitare che due utenti carichino contemporaneamente due file con lo stesso nome. Il file viene salvato in una cartella temporanea per poi essere spostato, a fine elaborazione, nella cartella *Articoli*. Viene controllato che nella cartella *articoli* non esista già un file con quel nome e se sì si avvisa l'utente con un messaggio e si provvede a cancellare i file creati:

```
'verifica se il file caricato esiste già
set FileObject=Server.CreateObject(
                              "Scripting.FileSystemObject")
p_FileEsistente="c:\inetpub\wwwroot\ifp\docenti
                              \articoli\" & p_subfileName1
if FileObject.fileExists(p_FileEsistente) then
   ' il file esiste e contiene un altro articolo. Devi
                                   scegliere un altro nome
   p_err=1
   FileObject.DeleteFile(p_subfileName)
set FileObject=nothing
```

Se il nome del file non è presente tra gli articoli pubblicati, si procede alla creazione della pagina asp che conterrà il testo dell'articolo, comprese le intestazioni, e si sposta il file nella cartella articoli. Il nome della pagina asp sarà uguale al nome del file di testo.

```
'sposta il file di testo nella cartella Articoli
FileObject.MoveFile p_subfileName,p_FileEsistente
set FileObject=nothing
'crea il  file asp che conterrà l articolo
p_NomeFile=p_subfileName1
p_NomeFile=replace(p_NomeFile,".txt",".asp")
set NewFileObject=Server.CreateObject(
                              "Scripting.FileSystemObject")
set NewFile= newFileObject.CreateTextFile("c:\inetpub\
              wwwroot\ifp\docenti\articoli\" & p_NomeFile)
newFile.WriteLine("<!--#include virtual=" & chr(34)
                    & "ifp/indexRiga.txt" & chr(34) & "-->")
newFile.WriteLine("<pre><b>" & p_TitoloArticolo
                                        & "</b><br><br>")
newFile.WriteLine("<!--#include virtual=" & chr(34) &
```

```
        "ifp/docenti/articoli/" & p_subfileName1 & chr(34)
                                                   & "-->")
newFile.WriteLine("</pre></body></html>")
newFile.close
```

Infine vengono salvate tutte le informazioni nella tabella *Articoli*, utilizzando in metodo *AddNew* dell'oggetto *RecordSet*.

CANCELLAZIONE ARTICOLI

Il docente può anche eliminare un suo articolo. Questo sarà rimosso dalla tabella *Articoli*, ma i file contenenti il testo (sia il file di testo che i files asp) saranno spostati nella cartella *\ifp\cestino* e vi resteranno finchè l'amministratore non svuoterà il cestino. Prima di accedere alla selezione degli articoli da cancellare, saranno richieste nuovamente *Username* e *Password* del docente dal form contenuto nella pagina *CancellaFormArticoli.asp*. Accertati i dati, verranno elencati tutti gli articoli pubblicati dal docente (pagina *Esegui_FormArticoli.asp*).

```
SELECT Articoli.IDarticolo, Articoli.CodiceCorso,
        Articoli.TitoloArticolo, Articoli.DataPubblicazione,
  Articoli.IndirizzoFile FROM Docenti INNER JOIN Articoli
  ON Docenti.IDdocente = Articoli.IDdocente GROUP BY
             Articoli.IDarticolo, Articoli.CodiceCorso,
        Articoli.TitoloArticolo, Articoli.DataPubblicazione,
               Articoli.IndirizzoFile, Docenti.UserName,
      Docenti.Password HAVING (((Docenti.UserName)=
       '" & p_username & "') AND ((Docenti.Password)=
                     '" & p_password & "')) ORDER BY
                        Articoli.DataPubblicazione DESC
```

Con questa interrogazione Sql si estraggono tutti gli articoli del docente (vedi la clausola *Having*) in ordine decrescente di data di pubblicazione. Vengono poi elencati tutti gli articoli in una tabella contenente nel primo campo un checkbox che ha la stessa funzione di quello usato nella pagina per la modifica/cancellazione dei corsi (se selezionato assume come valore l'*IDarticolo*)

```
<% While not tableSet.EOF %>
<tr><TR><TD>
<INPUT TYPE="checkbox" NAME="p_delete"
          VALUE="<%=tableSet.Fields(0).value%>">
</TD>
<%for x = 1 to (p_numerOfColumns-1)
  if x=4  then %>
    <Td> <a href=<%=Server.URLEncode(Cstr(
                    tableSet.Fields(4).value))%>>
      <%=Cstr(tableSet.Fields(4).value )%></a></Td>
  <%else%>
    <Td> <%=tableSet.Fields(x).value %></Td>
  <%end if
next
tableSet.MoveNext
wend
```

Tramite i checkbox è possibile selezionare gli articoli da cancellare e cliccando sul tasto del form *Cancella*, gli articoli selezionati si passa alla pagina *Esegui_Esegui_CancellaFormArticoli.asp.* La prima operazione da fare è determinare quanti record devono essere cancellati

```
NumeroRecordCancellare=Request.Form("p_delete").Count
```

Il ciclo successivo cancella i record degli articoli dalla tabella *Articoli* e tutti i file relativi ad essi (li sposta nella cartella *Cestino*):

```
for p_cancella=1 to NumeroRecordCancellare
p_ID_DaCancellare=Request.form("p_delete").item(
                                          p_cancella)
deleteSet.Open "select * from Articoli where
          IDarticolo= " & _ p_ID_DaCancellare,IFPDB,
       adOpenDynamic, adLockPessimistic, adCmdText
if not deleteSet.EOF then
p_IndirizzoFile="c:\inetpub\wwwroot\ifp\docenti\
              articoli\" & _ deleteSet("IndirizzoFile")
p_fileName="c:\inetpub\wwwroot\ifp\docenti\articoli\"
                          & _ deleteSet("NomeFile")
p_fileNameImode="c:\inetpub\wwwroot\ifp\docenti\
          articoli\" & _ deleteSet("IndirizzoFileImode")
p_IndirizzoFileCestino="c:\inetpub\wwwroot\ifp\
              cestino\" & _ deleteSet("IndirizzoFile")
p_fileNameCestino="c:\inetpub\wwwroot\ifp\cestino\"
                          & _ deleteSet("NomeFile")
p_fileNameImodeCestino="c:\inetpub\wwwroot\ifp\
          cestino\" & _  deleteSet("IndirizzoFileImode")
  fileDaSpostare.MoveFile p_IndirizzoFile,
                              p_IndirizzoFileCestino
  fileDaSpostare.MoveFile p_fileName,p_fileNameCestino
  fileDaSpostare.MoveFile p_fileNameImode,
                          p_fileNameImodeCestino
  deleteSet.delete
deleteSet.UpDate
end if
deleteSet.Close
next
```

INVIO COMUNICAZIONI

Dal form *InserimentoComunicazioni* si inseriscono tutti i dati relativi alla comunicazione da inviare agli studenti, compreso il nome del file di testo contenente il messaggio. La pagina asp *Esegui_InserimentoComunicazione* utilizza entrambi i componenti *Persits: AspUpload* e *AspEmail.* Per l'upload il proce-

dimento è molto simile a quello utilizzato nell'inserimento di articoli: l'applicazione viene bloccata per evitare che due utenti diversi carichino sul server due file con lo stesso nome; il file di testo viene salvato nella cartella *\ifp\docenti\articoli\temp* e dopo l'invio della e-mail, viene cancellato.
Prima di inviare il messaggio, viene creata una stringa contenente tutti gli indirizzi degli studenti iscritti al corso oggetto della comunicazione.

```
'estrai gli indirizzi dei destinatari
sqlIndirizzi="SELECT Studenti.email, StudentiIscritti.
   CodiceCorso FROM Studenti  INNER JOIN StudentiIscritti
                  ON Studenti.IDstudente = StudentiIscritti.
               IDstudenteIscritto WHERE (((StudentiIscritti.
                  CodiceCorso)= '" & p_codiceCorso & "'))"
set userSetIndirizzi=IFPDB.Execute(sqlIndirizzi)
'prepara la lista di indirizzi
p_indirizzo=""
while (not userSetIndirizzi.EOF)
p_Indirizzo=p_indirizzo & trim(
         userSetIndirizzi.fields.item("email")) & "," & " "
userSetIndirizzi.MoveNext
wend
set userSetIndirizzi=nothing
```

IL SERVIZIO I-MODE

i-mode è un servizio che permette di accedere ad Internet dal cellulare. Questa nuova tecnologia è stata lanciata in Giappone dalla NTT DoCoMo, che detiene tuttora il marchio. In Italia la prima compagnia telefonica ad introdurre servizi i-mode è stata Wind. La connessione i-mode è sempre attiva e il costo dipende (sarebbe più corretto dire dipenderà, dal momento che attualmente sono ancora in vigore le offerte promozionali) dai KB scambiati. Un cellulare dotato di servizi i-mode è in grado di interpretare il cHTLM (Compact HTLM), una versione compatta del linguaggio HTML. E' possibile leggere le specifiche dello standard cHTML del *World Wide Web Corsortium* (W3C) all'indirizzo *www.w3.org/TR /1998/NOTE-CompactHTML-19980209*. La DoCoMo, inoltre, ha messo a disposizione gratuitamente una guida in pdf all'indirizzo *www.imode.nl/imode/gfx /attachments/How%20to%20create%20an%20i-mode%20site.pdf* Dal cHTML sono escluse alcune funzioni: mappe di immagini, immagini JPEG, tabelle, frame, fogli di stile, immagini di sfondo; si dispone in una pagina di un solo font e un solo stile. Sono state introdotte le *accesskey:* è possibile associare un numero ad un link e selezionarlo da tastiera premendo il tasto corrispondente.
Ultimo particolare: una pagina cHTML può essere visualizzata sul computer da un qualsiasi browser e può essere ospitata su un comune host adibito ad ospitare siti WEB.

PASSIAMO ALLA PRATICA

L'Istituto di Formazione professionale vuole creare dei sevizi anche per gli studenti che posseggono un cellulare i-mode. I servizi previsti sono: consultazione dei corsi attivati (area accessibile a tutti), visione degli articoli, prenotazione esami (questi ultimi due solo per gli studenti registrati).

La homepage per i-mode

Analizziamo il contenuto della pagina *imode_index.asp*.

```
<%@ LANGUAGE="VBSCRIPT" %>
<!DOCTYPE HTML PUBLIC -//W3C
                     //DTD Compact HTML 1.0 Draft//EN>
<HTML> <head>
<TITLE>Istituto di formazione professionale.
                               Servizio i-mode</TITLE>
  <meta http-equiv="content-type" content=
                             "text/html;charset=x-sjis">
  <meta name="CHTML" content="yes">
</head>
<BODY TEXT="#000000" BGCOLOR="99CCFF">
<CENTER>Istituto di formazione professionale.
                       Servizio <I>i</I>-mode home
</CENTER>
<hr noshade size="1">
<marquee behavior="scroll" direction="left"
                              width="150" loop="16">
Istituto di formazione professionale - servizio i-mode
</marquee>
<hr noshade size="1">
<a accesskey="1" href="http://localhost/ifp/corsi
              /imode_IndiceCorsi.asp">&#63879;</a>
                                   Indice corsi<br />
<a accesskey="2" href="http://localhost/ifp/docenti
        /Articoli/imode_FormArticoli.asp">&#63880;</a>
                                Indice Articoli<br />
<a accesskey="3" href="http://localhost/ifp/studenti/
         imode_FormPrenotazioni.asp">&#63881;</a>
                             Prenotazione Esame<br />
<a accesskey="4" href=mailto:xxxx.yyyyy@tiscali.it>
                                   [4]</a>Scrivi<br />
<a accesskey="5" href=tel:+39123456>[5]</a>
                                       Telefona<br />
<hr noshade size="1"> </body>
</html>
```

Naturalmente, il contenuto della pagina deve essere ridotto all'osso. Rispetto all'HTML le uniche differenze che si notano sono le AccessKey, che permettono di selezionare il collegamento premendo il tasto corrispondente, e la possibilità di comporre un numero di telefono attraverso un link. Il codice *#63879* farà apparire il numerino *1* incassato in un quadratino bianco in corrispondenza del link *Indice*

www.w3.org/TR/1998/NOTE-CompactHTML-19980209
specifiche dello standard cHTML del W3C

www.imode.nl/imode/gfx/attachments/How%20to%20create%20an%20i-mode%20site.pdf
guida in pdf su come costruire siti i-mode compatibili

***Fig. 1:** Il menu principale del servizio*

corsi (così per il *2* e il *3*); per i link con accesskey uguale a *4* e *5* viene semplicemente visualizzato il numerino racchiuso tra parentesi quadre. Con il link *4* richiama il programma di gestione della posta per inviare e-mail all'indirizzo specificato. Con il link *5* è possibile invece comporre il numero di telefono indicato e inoltrare la chiamata. Questa funzione non poteva mancare dato che abbiamo a che fare con dei telefoni cellulari! Chi non possiede un cellulare dotato di servizio i-mode, può testare la propria pagina cHTML con un simulatore online consultabile liberamente all'indirizzo *www.wapag.com/en/simulation-check*. Se si possiede IIS o PWS (sono comunque indispensabili per testare le pagine asp) non è necessario pubblicare preventivamente la propria pagina. Basta avviare il programma di Server Web e sostituire a localhost il proprio indirizzo IP. Per esempio se il proprio IP è *62.10.53.41* si dovrà inserire nel simulatore l'indirizzo completo: *http://62.10.53.41/ifp/imode_index.asp*.

NOTA

Chi non possiede un telefono cellulare dotato di servizio i-mode, può testare la propria pagina cHTML con un simulatore online consultabile liberamente all'indirizzo www.wapag.com/en/simulation-check

Form di prenotazione esami e consultazione articoli

I due form sono identici e richiedono l'inserimento di Username e Password:

```
<form action="Imode_Prenotazioni.asp" method="post">
  Username: <input type="text" name="p_username">
                                              <br />
  Password: <input type="password" name="p_
                                   password"><br />
  <input type="submit" value="Entra">
</form>
```

Anche le pagine collegate a questi due form sono molto simili ed elencano i corsi a cui uno studente è iscritto. Ricordiamoci che i tag *<table>* non sono consentiti

```
set IFPDB=Server.CreateObject("ADODB.Connection")
IFPDB.Open "IstitutoIFP"
set tableSet=Server.CreateObject("ADODB.Recordset")
tableSet.Open "SELECT Studenti.IDstudente,
         Studenti.UserName, StudentiIscritti.IDcorso,
 StudentiIscritti.CodiceCorso, Corsi.DataEsame FROM Corsi
    INNER JOIN (Studenti INNER JOIN StudentiIscritti ON
 Studenti.IDstudente = StudentiIscritti.IDstudenteIscritto)
      ON Corsi.IDcorso = StudentiIscritti.IDcorso WHERE
                            (((Studenti.IDstudente) =
                       " & p_IDutente & "))", IFPDB %>
Corsi ai quali sei iscritto
<hr noshade size="1">
<P>
<% While not tableSet.EOF %>
Corso  <A HREF="Imode_Esegui_FormPrenotazioni.asp?
          p_CodiceCorso=<%Response.Write tableSet(
      "CodiceCorso")%>;p_IDutente=<%Response.Write
                        tableSet("IDstudente")%>">
<%=tableSet.Fields(3).value%></A>
Data Esame <%=Cdate(tableSet.Fields(
                               "DataEsame").value)%>
<br /><% tableSet.MoveNext
wend
```

Sul sito sono disponibili per il test diversi modelli di cellulari.

Nelle pagine i-mode non sono stati utilizzati dei cookies. I parametri passano attraverso i collegamenti, pertanto bisogna evitare di trasmettere informazioni riservate come *Username* e *Password;* per identificare lo studente è stato passato come parametro il suo *ID*. Per la prenotazione la procedura è del tutto simile a quella utilizzata in precedenza.
Per gli articoli, viene visualizzato l'elenco di quelli relativi al corso selezionato:

```
Indice Articoli del corso <%=p_CodiceCorso%><br><br>
<% While not tableSet.EOF  ' titolo %>
<a href=<%=Server.URLEncode(Cstr(
                          tableSet.Fields(6).value))%>>
  <%=Cstr(tableSet.Fields(3).value )%></a><br />
<% tableSet.MoveNext
wend
```

Dalla tabella *Articoli* si preleva l'indirizzo del file formattato per l'i-mode e il cui percorso era stato salvato in un campo della tabella al momento dell'inserimento. Questo sarà il contenuto tipico di un file contenente un articolo formattato:

```
<!--#include virtual="ifp/ImodeTop.txt"-->
<b>Articolo su Imode</b><br />
<!--#include virtual="ifp/docenti/articoli
                              /ArticoloImode.txt"-->
<!--#include virtual="ifp/ImodeTop.txt"-->
```

mentre il contenuto del file *ImodeTop.txt* contiene la solita intestazione e un'immagine gif collegata alla *HomePage*

```
<IMG SRC="http://localhost/ifp/images/home.gif"
                    HEIGHT=26 WIDTH=26 BORDER=0>
</A>Istituto di formazione professionale.
                 Servizio <I>i</I>-mode Indice Articoli
```

CONCLUSIONI

Data la scarsa quantità di memoria dei telefoni cellulari e dal momento che chi naviga con i-mode paga in base ai KB scambiati, sarà bene progettare le pagine in modo tale che non superino i 5 KB, limitando le immagini e concentrando i contenuti. I nostri siti perderanno in "immagine", ma l'obiettivo in questo caso è quello di rendere dei servizi che possano essere utilizzati in modo proficuo.

Patrizia Martemucci

Web Service Enhancements 2.0

Instant Messenger con WSE 2

In casa Microsoft è in atto un'evoluzione verso una nuova piattaforma denominata Indigo. Ma tra i WS attuali e Indigo c'è WSE 2: scopriamo come utilizzarlo e con quali vantaggi

Far comunicare sistemi diversi, piattaforme diverse (magari distanti fisicamente), è stata una delle maggiori problematiche risolte dai servizi web. Un servizio Web altro non è che una porzione di codice eseguibile, dotato di una interfaccia standardizzata che lo rende accessibile via Web, tramite protocollo HTTP. In .NET, un Web Service ha la necessità di un web server. Seppur abbastanza nuova, questa tecnologia ha subito mostrato alcuni limiti. Primo tra tutti il protocollo di comunicazione. Sebbene HTTP sia estremamente diffuso, non sempre è l'ideale per la risoluzione di alcune problematiche. HTTP infatti si basa su un meccanismo di richiesta/risposta: il client invia una richiesta e resta in attesa di una risposta. Questa caratteristica lo rende inappropriato, ad esempio, in situazioni in cui si necessita di una comunicazione punto-punto o ancora in situazioni in cui ad una richiesta corrispondono risposte multiple. I *Web Service Enhancements* (WSE), ormai giunte alla seconda versione, alcune lacune dei servizi web tradizionali. Primo tra tutti il protocollo di comunicazione non più vincolato all'HTTP. La prima importante conseguenza di questa modifica è che, utilizzando WSE 2, non c'è più la necessità di avere un server WEB che fornisca il servizio. Possiamo quindi realizzare applicazioni che scambiano messaggi direttamente da un punto all'altro di una rete senza che ci sia un intermediario. Questo è esattamente quello che vedremo in questo articolo. Un altro vantaggio molto importante di questa modifica è che cade anche il vincolo intrinseco del protocollo HTTP: richiesta/risposta. Vedremo infatti che WSE ci consente di creare dei messaggi SOAP la cui risposta è magari indirizzata ad altri destinatari, e non a chi ha fatto la richiesta specifica.

Fig. 1: *L'instant messenger che realizzeremo*

WSE 2.0

Web Service Enhancements, in sostanza, cosa è? Cito dalla documentazione ufficiale: WSE è un motore per applicare avanzati protocolli per servizi wse ai messaggi SOAP. Questo potrebbe anche richiedere la trasformazione del corpo del messaggio SOAP. È bene chiarire subito che, come evidente nella frase appena citata, WSE non è qualcosa di molto diverso dai servizi Web tradizionali, ma un loro potenziamento. Quello che abbiamo già imparato sui servizi Web resta sostanzialmente valido. Alla base c'è sempre un messaggio SOAP che viaggia, c'è sempre un mezzo di comunicazione che lo fa viaggiare (il protocollo) e l'esigenza di base è sempre la stessa: far comunicare dei sistemi. WSE estende i servizi web tradizionali portandoli in un'ottica SOA *(Service Oriented Architecture)*. Le maggiori novità apportate da WSE sono:

- **Sicurezza:** l'unico modo per rendere sicuri dei tradizionali servizi Web è quello di usare SSL. WSE invece implementa le specifiche WS-Security. Tali specifiche sanciscono il modo in cui garantire che un messaggio SOAP sia integro (quindi non sia stato compromesso durante il trasporto), sia confidenziale e, soprattutto, sia

autentico. Immaginate cosa potrebbe succedere se un nostro messaggio che rappresenta un ordine venisse intercettato e magari modificato! WS-Security fornisce inoltre il meccanismo che consente di associare le credenziali direttamente nel messaggio, magari cifrandole.

- **Allegati:** la specifica *WS-Attachments* descrive il modo in cui poter inserire degli allegati non serializzati in XML, nel messaggio SOAP.

- **Routing:** il routing di un messaggio SOAP consiste, in breve, nella possibilità di trasferire un messaggio SOAP da un punto ad un altro, attraverso diversi intermediari. I meccanismi per poter implementare il routing dei messaggi sono definite nelle specifiche *WS-Routing*.

- **SOAP Messaging:** è forse la parte chiave di tutta l'architettura WSE. In definitiva, fornisce tutte le funzionalità che consentono l'invio del messaggio SOAP utilizzando protocolli diversi dall'HTTP, nonché la possibilità di gestire l'invio punto – punto ecc.

Il punto di partenza per approfondire l'argomento è http://msdn.microsoft.com/webservices/. Oltre ai tradizionali servizi web, ci sono tutte le informazioni relative a WSE

ANATOMIA DI UN MESSAGGIO SOAP

Dopo questo accenno abbastanza generale ed introduttivo a WSE 2.0, muniamoci dell'occorrente e iniziamo a lavorare. Per farlo, dobbiamo innanzitutto scaricare WSE 2.0 dal sito Microsoft (vedi box laterale) ed installarlo. L'operazione richiede pochi minuti e non necessita del riavvio del sistema. Completata l'installazione di WSE 2.0, iniziamo immediatamente a sperimentare qualcosa, magari "guardando" direttamente un messaggio SOAP. Apriamo quindi Visual Studio .NET 2003 e creiamo un nuovo progetto *(WindowsForm)* scegliendo come linguaggio C#; diamo quindi al progetto il nome *InstantMessenger.* Oltre a creare il nostro progetto, Visual Studio creerà per noi anche la soluzione (chiamata anch'essa *InstantMessenger*) in cui andremo ad inserire gli altri nostri progetti. Abbandoniamo per un po' il progetto *Windows Form* e creiamo un nuovo progetto. Questa volta scegliamo *Applicazione Console* che chiameremo *SoapMessage*. Per prima cosa, aggiungiamo un riferimento a WSE cliccando con il tasto destro del mouse sulla cartella *References,* selezionando la voce *Aggiungi riferimento* e selezionando dall'elenco *Microsoft.Web.Service2.dll*. Con le direttive *using,* predisponiamoci a lavorare con WSE2:

```
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Addressing;
```

Fig. 2: Schema dell'architettura WSE

Lo scopo che ci prefiggiamo in questo primo esperimento è quello di vedere in chiaro il nostro messaggio SOAP. Nel metodo *Main* del nostro progetto, iniziamo a costruire un messaggio di esempio:

```
[STAThread]
static void Main(string[] args){
  SoapEnvelope messaggio = new SoapEnvelope();
  messaggio.CreateBody();
  messaggio.Body.AppendChild(
                messaggio.CreateElement("Nodo1"));
  messaggio.Body.FirstChild.AppendChild(
                messaggio.CreateTextNode("Valore1"));
  messaggio.Body.AppendChild(
                messaggio.CreateElement("Nodo2"));
  messaggio.Body.ChildNodes[1].AppendChild(
                messaggio.CreateTextNode("Valore2"));
```

SoapEnvelope è la nostra "busta", quella che conterrà l'intero messaggio. Questa classe deriva da *XMLDocument,* quindi il nostro messaggio può essere costruito come se fosse un normale file XML (proprio come nell'esempio). In alternativa, possiamo lasciare al *SoapEnvelope* il compito di creare il body del messaggio passandogli direttamente un oggetto. *SoapEnvelope* infatti implementa il motore di serializzazione XML che serializzarà per noi l'oggetto passato e creerà il body del messaggio:

```
messaggio.SetBodyObject(mioOggetto);
```

Questo sarà indubbiamente il sistema maggiormente utilizzato, ma ora ci interessa capire cosa fa esattamente l'Envelope. Una volta creato il corpo messaggio, inseriamo le informazioni sul recapito dello stesso:

```
messaggio.Context.Addressing.Action = new Action(
                    "urn:ProvaSoapMessage");
messaggio.Context.Addressing.From = new From(new
 EndpointReference(new Uri("soap.tcp: //ioProgrammo:
                    9000/ProvaSOAP")));
messaggio.Context.Addressing.To = new To(new Uri
        ("soap.tcp://iPprogrammo:9001/ProvaSOAP"));
```

Per prima cosa, dichiariamo una *Action* per il nostro messaggio. Aggiungiamo poi le informazioni sul recapito vero e proprio (il *from* ed il *to*). A questo punto il nostro messaggio è pronto per essere inviato con l'istruzione:

```
SoapSender ss = new SoapSender(new Uri(.........));
ss.Send(messaggio);
```

Ma quello che ci interessa vedere è il contenuto del messaggio, una volta che è stato processato dalla Pipeline (vedi Fig. 2). Procediamo al processing:

```
Pipeline pl = new Pipeline();
```

```
pl.ProcessOutputMessage(messaggio);
```

e al salvataggio del messaggio stesso direttamente sull'hard disk:

```
messaggio.Save(Environment.CurrentDirectory+
                              "\\messaggio.xml");
```

Una volta eseguito il codice, nella cartella di esecuzione del programma ci troveremo un nuovo file XML che conterrà il nostro messaggio. Come visibile dall'immagine in Fig. 3, il messaggio è costituito da un XML contenente sia le informazioni relative al *body* che abbiamo realizzato manualmente, sia le informazioni aggiuntive relative all'*action* e all'*addressing*. Da notare anche due nodi abbastanza importanti:

```
<wsu:Created>2004-09-01T23:17:42Z</wsu:Created>
<wsu:Expires>2004-09-01T23:22:42Z</wsu:Expires>
```

che identificano il momento di creazione del messaggio stesso nonché la sua scadenza.

IL NOSTRO INSTANT MESSENGER

Ora che abbiamo visto com'è fatto un messaggio e come si crea, realizziamo un piccolo Instant Messenger. Quello che realizzeremo è sostanzialmente un piccolo software che consente lo scambio dei messaggi tra due client. L'esempio, sebbene sia abbastanza semplice nella sua logica, evidenzia due aspetti importanti di WSE:

1. la comunicazione avviene punto – punto, quindi senza la necessità di avere un server web che ospiti il Web Service;
2. la comunicazione avviene sfruttando un protocollo diverso dall'HTTP.

Torniamo al nostro progetto *Windows Form* creato precedentemente e, seguendo la stessa procedura applicata al progetto *SoapMessage,* aggiungiamo due riferimenti: *Microsoft.Web.Service2 .dll* e *System. Web.dll.* L'utilizzo di *System.Web .dll* è indispensabile in quanto una delle classi che utilizzeremo *(SoapReceiver),* implementa internamente l'interfaccia *IHttpHandler* contenuta appunto in *System.Web.dll.* Con la direttiva *using,* definiamo subito quali namespace utilizzare nella nostra applicazione:

```
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Addressing;
using Microsoft.Web.Services2.Messaging;
```

che, come abbiamo visto nell'esempio precedente, ci consentono di definire il messaggio e gli indirizzi a cui lo stesso deve essere inviato. Prepariamo quindi un form che assomigli a quello di Fig. 1 e definiamone il comportamento. Come prima cosa dobbiamo costruire il messaggio SOAP da inviare. Lo facciamo sull'evento *click* del pulsante *Invia*:

```
private void btnInviaMessaggio_Click(object sender,
                              System.EventArgs e) {
  SoapEnvelope messaggio = new SoapEnvelope();
  messaggio.SetBodyObject(tbxMessaggioDaInviare.Text);
```

A differenza dell'esempio *SoapMessage* visto in precedenza, questa volta lasciamo che sia il motore WSE a creare per noi il body del messaggio. Al metodo *SetBodyObject* del nostro envelope, passiamo la *textBox* contenente il messaggio che abbiamo scritto. Il motore di serializzazione si occuperà di creare gli opportuni nodi XML al fine di realizzare un XML valido. Preparato il messaggio, dobbiamo indirizzarlo a qualcuno. Di questo se ne occupa la sezione *Addressing* del Contex del nostro *SoapEnvelope* (nel nostro caso è messaggio).

Fig. 3: **Messaggio SOAP processato dalla Pipeline**

Definiamo quindi una *Action:*

```
messaggio.Context.Addressing.Action = new
                              Action("urn:IM");
```

successivamente, indichiamo gli indirizzi del mittente e del destinatario. Queste informazioni verranno inserite nell'*envelope* e ci consentiranno di scambiare il messaggio SOAP che abbiamo precedentemente creato.

```
messaggio.Context.Addressing.From = new From(
                    new Uri(tbxInAscoltoSu.Text));
messaggio.Context.Addressing.To = new To(new Uri(
                              tbxTo.Text));
```

Giunti a questo punto, si potrebbe pensare che le informazioni sull'addressing siano state già inserite nel messaggio. Basterà però salvare il messaggio come abbiamo fatto in precedenza *(messaggio.Save (path))* per rendersi conto che questo non è vero. Riprendiamo l'immagine di Fig. 2. Noi siamo ancora sotto il cilindro della Pipeline! Per far si che le informazioni aggiuntive che abbiamo preparato siano inserite nel messaggio stesso, dobbiamo o forzare la Pipeline (come abbiamo fatto nel precedente messaggio), oppure inviare il messaggio stesso:

Il download di WSE 2.0 (ora giunto alla SP1) è disponibile qui:
www.microsoft.com/downloads/details.aspx?FamilyId=FC5F06C5-821F-41D3-A4FE-6C7B56423841&displaylang=en

Paolo Pialorsi, di DevLeap
www.devleap.com
ha messo on line quattro interessanti video (in inglese) su WSE 2. I contenuti sono raggiungibili qui:
www.devleap.com/SchedaArticolo.aspx?IdArticolo=10730

```
SoapSender sSender = new SoapSender(new
                                     Uri(tbxTo.Text));
sSender.Send(messaggio);
```

Sarà il *SoapSender* a far passare il messaggio dalla Pipeline che "plugghera" in esso tutte le informazioni extra tipiche di WSE. Se salviamo ora il messaggio, ci troveremo d'avanti qualcosa di molto simile alla Fig. 4 in cui è ben visibile il messaggio scritto, inserito in un nodo XML dal *SetBodyObject*. Completata la parte relativa all'invio del messaggio, realizziamo quella per la ricezione. Definiamo una classe *Ricevitore* che avrà il compito di ricevere il messaggio e di mostrarlo nell'apposita area. *Ricevitore* è una classe che deriva dalla *SoapReceiver*, di cui fa l'override del metodo *Receive*.

```
public class Ricevitore : SoapReceiver {
  public RichTextBox MessaggioRicevuto;
  public TextBox To;
  protected override void Receive(SoapEnvelope envelope) {
MessaggioRicevuto.AppendText(envelope.GetBodyObject(
                       typeof(String)).ToString()+"\r\n");
    To.Text = envelope.Context.Addressing.From
                          .Address.Value.AbsoluteUri; }}
```

Receive prende come argomento un *SoapEnvelope* (nel caso specifico, quello che abbiamo inviato con *sSender.Send()*). Dall'envelope ricevuto, estraiamo due informazioni:

1. Il messaggio vero e proprio; quello che abbiamo scritto nella textbox *tbxMessaggioDaInviare*
2. Il mittente del messaggio che diventerà il destinatario a cui inviare la risposta.

Manca ancora qualcosa. Stiamo realizzando una comunicazione punto – punto ma… non abbiamo ancora definito i due punti della comunicazione! Demandiamo il compito all'evento *click* del bottone *btnInAscoltoSu* che, tra le altre operazioni di relativa importanza, creerà un *EndPoint:*

```
SoapReceivers.Add(new EndpointReference(new
              Uri(tbxInAscoltoSu.Text)), ricevitore);
```

Che sarà in pratica, il "punto" della comunicazione punto – punto. Il metodo *Add* di *SoapReceiver* ha quattro overload. Noi utilizzeremo il quarto che si aspetta due parametri:

1. Un *EndPoint* caratterizzato da un *Uri*. Nel nostro caso, l'*Uri* è quello che abbiamo dichiarato nella textbox *tbxInAscoltoSu*.
2. Il secondo parametro è un riferimento all'oggetto che si occuperà di ricevere il nostro messaggio. Nel nostro caso, passeremo una istanza della classe *Ricevitore*.

Su Windows XP, se avete il firewall abilitato, non dimenticate di aprire le porte utilizzate per la comunicazione punto – punto. Nella SP2 di Windows XP invece, il firewall integrato vi chiederà cosa fare. Autorizzate il programma per fare i test.

Abbiamo praticamente detto al nostro programma che deve aspettarsi i dati all'indirizzo che abbiamo specificato nella *tbxInAscoltoSu* e che, chi riceverà i messaggi è la nostra classe *ricevitore*.

NON CI RESTA CHE PARLARE

Abbiamo il nostro sistema di Instant Messaging personale, ora non ci resta che provarlo! Avviamo due volte il programma e predisponiamolo alla comunicazione. Nella prima finestra, dichiariamo un indirizzo, completo di porta, rispetto al quale ci metteremo in ascolto. Ad esempio *soap.tcp://localhost:9000/IM* e clicchiamo sul pulsante *In Ascolto*. Nella seconda finestra, definiamo una porta diversa, ad esempio *soap.tcp:// localhost:9001/IM* e, anche qui, clicchiamo sul relativo pulsante. Torniamo nella prima finestra, inseriamo l'indirizzo di destinazione nella casella *To* (in questo caso sarà quello in cui abbiamo messo in ascolto la seconda finestra), scriviamo qualcosa nell'apposita area e premiamo *Invia*. Il nostro messaggio sarà inviato all'indirizzo del destinatario e, oltre ad apparire nell'area dedicata ai messaggi, verrà compilato automaticamente anche il campo *To*. Se il test va a buon fine, possiamo utilizzare il nostro Messenger su due PC diversi. La procedura è identica tranne che per due fattori: l'indirizzo di ascolto (localhost) deve essere sostituito con l'IP pubblico del PC su cui è in esecuzione il programma. Per recuperarlo, avviamo la shell (*start/esegui /cmd*) e digitiamo il comando *ipconfig*. Se inoltre abbiamo un firewall attivo (Windows XP lo ha), dobbiamo aprire la porta scelta per la ricezione dei messaggi (nel caso dell'esempio è la porta 9000).

CONCLUSIONI

In questo articolo abbiamo visto alcune delle caratteristiche generali di Web Service Enhancements 2.0. L'esempio dell'Instant Messenger ha lo scopo di mostrare solo una piccola parte delle potenzialità di questa architettura nonché una delle differenze sostanziali rispetto ai tradizionali servizi Web: l'assenza del server Web. L'utilizzo di WSE in ambienti professionali può portare enormi vantaggi sia in termini economici (tempo risparmiato nella scrittura del codice, server in meno da gestire e configurare ecc.), sia in termini di funzionalità e di sicurezza. Un personale consiglio è quello di approfondire l'argomento, sia per coglierne già da oggi i vantaggi, sia per essere pronti in futuro (nemmeno troppo lontano) ad Indigo. Nel prossimo numero amplieremo il nostro Instant Messenger implementando anche l'invio di file in allegato. Buon lavoro.

Michele Locuratolo

Miglioriamo le nostre applicazioni con il VB Power Pack

VB Power Pack

VB.NET permette la realizzazione di applicazioni molto interessanti, ma per un'interfaccia grafica "avanzata" bisogna ricorrere a dei componenti esterni: proviamo sette nuovi controlli gratuiti

La ricchezza della libreria .NET e la completezza del linguaggio permettono finalmente di realizzare qualsiasi tipo di applicazione, anche quelle più sofisticate, dotate di funzionalità molto difficili da realizzare in Visual Basic 6, come ad esempio la gestione del multi-threading. La parte forse più debole, però, rimane quella dell'interfaccia utente. La Microsoft e altri produttori ci hanno abituato ad interfacce sofisticate, con *TaskPane* collassabili, finestre di notifica *(Toast)* alla MSN Messenger, sfondi colorati, barre e bottoni grafici, etc…
.NET non dispone di questo genere di controlli, ma ha tutte le potenzialità per realizzarli. Il Visual Basic Power Pack è un insieme di controlli grafici scritti in VB.NET, che possono però essere utilizzati in qualsiasi linguaggio .NET. In questo articolo vedremo come utilizzarli per migliorare l'interfaccia utente della nostra applicazione.

NOTA

VB POWER PACK È GRATIS?
Si, VB Power Pack è completamente gratuito, sia per l'utilizzo personale, sia per lo sviluppo di applicazioni commerciali.

FileSystemMonitor

Scopo dell'applicazione che verrà realizzata è quello di permettere di controllare file e cartelle, anche più di una contemporaneamente, e di notificare l'utente quando si verifica un cambiamento. Per funzionare sfrutta la classe *FileSystemWatcher* di .NET, che funziona solo su sistemi operativi della famiglia Windows NT, quindi: NT4, 2000, XP e 2003. L'interfaccia utente dell'applicazione è mostrata in Fig. 1. Come si vede in figura, è in corso il monitoraggio del file *"prova.txt"* e della cartella *"C:\Downloads"*. Si è scelto di utilizzare un'interfaccia di tipo *"tabbed"*, come fanno il Visual Studio.NET, browser come FireFox o Maxthon, e programmi come UltraEdit, ecc… Questo approccio è più user friendly dei normali approcci SDI e MDI, supportati nativamente dal framework. Per implementare questo tipo di interfaccia si è scelto un *TabControl* impostato in modalità *Flat*.

Fig. 1: Interfaccia utente dell'applicazione FileSystemMonitor

INTRODUZIONE AI CONTROLLI DEL POWER PACK

Il Power Pack comprende i seguenti controlli:

- **BlendPanel** – permette di inserire sfondi sfumati sulle nostre form.
- **UtilityToolbar** – è una toolbar molto simile a quella di Internet Explorer.
- **ImageButton** – è un pulsante che permette di mostrare immagini su sfondo trasparente.
- **NotificationWindow** – è la finestrella di notifica (alla MSN Messenger).
- **TaskPane** – è molto simile alla barra "collassabile" presente nell'*Esplora Risorse* di Windows XP.
- **FolderViewer** – è un TreeView che mostra le

Fig. 2: Selezione dei componenti del Power Pack

Directory dell'hard disk.

- **FileViewer** – è una ListView ottimizzata per mostrare file.

Il Power Pack è liberamente scaricabile e ridistribuibile (sia la dll, sia i sorgenti) con una licenza di tipo *Shared Source*. Per semplificare l'utilizzo dei controlli, conviene aggiungerli alla Toolbox del Visual Studio, premendo il tasto destro sulla Toolbox stessa e selezionando *Aggiungi/Rimuovi (Add Remove Items…)*. Si possono vedere i vari componenti da aggiungere in Fig. 2: tutti i componenti appartengono al namespace *VbPowerPack*.
A questo punto, i controlli possono essere trascinati sulle nostre form e utilizzati come se fossero componenti di sistema. Per comodità io li ho inseriti in una sezione apposta della Toolbox (Fig. 3).

LA FORM PRINCIPALE

Di seguito trovate un elenco degli elementi che andranno a comporre la form.

- Un controllo **MainMenu** di .NET *(MainMenu1)*
- Un controllo **UtilityToolbar** del VB PowerPack *(UtTMain)*, con *Appearance=Flat* e *Dock=Top*
- Un controllo **TabControl** di .NET *(TabCMain)*, con *Anchor* su tutti e quattro i lati, e grande come tutta la Form esclusi la *Toolbar* e la *StatusBar*
- Un controllo **StatusBar** di .NET *(StatusBar1)*, con *Dock=Bottom*
- Un controllo **NotificationWindow** del VB PowerPack *(NotificationWindow1)*, con *ShowStyle= Fade, CloseButton=True*, e con i colori di sfondo che più ci piacciono
- Un controllo **ContextMenu** di .NET *(ContextMenu1)*
- Un controllo **NotifyIcon** di .NET *(NotifyIcon1)*, con *ContextMenu=ContextMenu1*, l'icona *Normal.ICO* e *Visible=False*

Selezionando la *UtilityToolbar* nel pannello delle proprietà, e premendo il tasto in corrispondenza di *Buttons* è possibile configurare i bottoni presenti nella Toolbar, come in Fig. 4. La Toolbar del VB Power Pack è ottimizzata per contenere quei particolari bottoni. Può essere estesa anche con altri pulsanti, ma purtroppo non possono avere icone, in quanto il controllo non lo permette. A questo punto, associamo al tasto *Search* della Toolbar e al menu *File->New*, l'handler degli eventi che ci permetterà di aggiungere i file da monitorare. Tralasciando per ora la selezione del file o della cartella, occupiamoci di creare la parte grafica che mostrerà il log delle modifiche e che preparerà il *FileSystemWatcher:*

Fig. 4: Configurazione Toolbar

```
Dim frmCh As New FormChoose, wPath As String,
                              wFilter As String, tp As TabPage
If frmCh.ShowDialog() = DialogResult.OK Then
wPath = frmCh.Path
    wFilter = frmCh.Filter
If frmCh.IsFile Then
tp = New TabPage(wPath)
Else
tp = New TabPage(wFilter)
End If
tp.Dock = DockStyle.Fill
Dim t As New TextBox
t.ScrollBars = ScrollBars.Both
t.WordWrap = False
t.Multiline = True
t.ReadOnly = True
t.Dock = DockStyle.Fill
tp.Controls.Add(t)
Me.TabCMain.TabPages.Add(tp)
Dim watcher As FileSystemWatcher = New
                                        FileSystemWatcher
watcher.Path = wPath
watcher.Filter = wFilter
watcher.NotifyFilter = CType(frmCh.NotifyFilt, NotifyFilters)
watcher.EnableRaisingEvents = True
AddHandler watcher.Changed, New FileSystemEventHandler
                              (AddressOf Me.OnDiskChanged)
Dim fw As FileWatcher
fw.Watcher = watcher
fw.TabPage = x
Watchers.Add(fw)
End If
Me.StatusBar1.Text = "Currently monitoring " +
              Watchers.Count.ToString + " file(s)/dir(s)..."
```

In cui *Watchers* è una ArrayList che ci permetterà di recuperare a runtime le informazioni che ci servono. *Watchers* contiene la seguente struttura:

```
Private Structure FileWatcher
Public Watcher As FileSystemWatcher
Public TabPage As TabPage
End Structure
```

Per intercettare gli eventi generati dal *FileSystemWatcher* dobbiamo preparare il suo event handler. Gli eventi generati dal *FileSystemWatcher* vengono generati in un altro thread e quindi, per poter inte-

Fig. 3: Toolbar del Visual Studio modificata

VB POWER PACK È UN PRODOTTO SUPPORTATO?

No, VB Power Pack è un prodotto rilasciato con licenza *Shared Source*, di cui sono disponibili i sorgenti, e che NON è ufficialmente supportato da Microsoft. Però sul Workspace *GotDotNet* che lo ospita potete trovare tutte le risorse e gli aiuti necessari.

Fig. 5: Icona del programma nella Notification Area

VB POWER PACK GIRA ANCHE CON C#?

Naturalmente sì. I controlli contenuti nel VB Power Pack possono essere utilizzati in C# e in tutti gli altri linguaggi .NET che supportano le Windows Forms.

ragire con l'interfaccia utente, dobbiamo utilizzare l'istruzione *Invoke* sulla nostra form, altrimenti si potrebbero avere problemi molto seri, fino al blocco dell'applicazione. Nel nostro caso, il controllo *NotificationWindow* non funziona se richiamato da un altro thread, e rimane dello "sporco" sullo schermo. Dobbiamo quindi, prima di tutto, preparare un *delegate* che ci permetterà di trasferire il controllo al thread della form principale:

```
Private Delegate Sub MarshalFSWEventToUIThread(
                        ByVal source As Object, ByVal e As
                                        FileSystemEventArgs)
```

Il *delegate* ha la stessa firma dell'evento generato dal *FileSystemWatcher*, cosa che ci permetterà di passare gli stessi parametri al thread che gestisce l'interfaccia utente. Possiamo quindi preparare i due event handler: uno che verrà eseguito nel thread secondario del *FileSystemWatcher*, e uno a cui verrà trasferito il controllo sul thread principale.

```
Private Sub OnDiskChanged(ByVal source As Object,
                        ByVal e As FileSystemEventArgs)
Dim args() As Object = {source, e}
Me.Invoke(New MarshalFSWEventToUIThread(
                        AddressOf OnDiskChanged_UI), args)
End Sub
Private Sub OnDiskChanged_UI(ByVal source As
                Object, ByVal e As FileSystemEventArgs)
For Each watch As FileWatcher In Watchers
If watch.Watcher Is CType(source, FileSystemWatcher) Then
NWLastMessage = DateTime.Now.ToShortTimeString +
                " " + Path.GetFileName(e.FullPath) + " " +
                                        e.ChangeType.ToString
CType(watch.TabPage.Controls(0), TextBox).Text +=
                                NWLastMessage + vbCrLf
If Me.ShowToast Then
If NWForm Is Nothing OrElse NWForm.Visible = False Then
NWForm = Me.NotificationWindow1.Notify(NWLastMessage)
End If
End If
If Me.NotifyIcon1.Visible = True Then
   Me.NotifyIcon1.Icon = New Icon(Me.GetType,
                                        "Changed.ICO")
End If
Exit For
End If
Next
End Sub
```

Nell'event handler che si occupa di gestire l'interfaccia utente, viene ricavato quale *Watcher* della collezione ha generato l'evento, viene impostato l'ultimo messaggio (nel caso lo si voglia rivedere), viene scritto il messaggio nel TextBox e, se si vogliono visualizzare i messaggi nella *NotificationWindow*, viene mostrato il *Toast*. Si noti che il messaggio viene mostrato solo se la finestra di un messaggio precedente è già sparita, altrimenti si correrebbe il rischio di riempire lo schermo di finestrelle (mi è capitato durante lo sviluppo…). Inoltre, se la *NotifyIcon* è visibile, viene cambiata con un'altra icona, per segnalare all'utente la modifica.

Fig. 6: Form per selezionare file o cartelle

MINIMIZZAZIONE NELLA NOTIFICATION AREA

Il programma consente anche di essere minimizzato nella *Notification Area* (vicino all'orologio, come in Fig. 5). Per fare questo si usa il seguente codice:

```
Private Sub FormMain_LocationChanged(ByVal sender
        As Object, ByVal e As System.EventArgs) Handles
                                MyBase.LocationChanged
If Me.WindowState = FormWindowState.Minimized And
                        Me.MinimizeInNotifyArea Then
Me.ShowInTaskbar = False
      Me.NotifyIcon1.Visible = True
End If
End Sub
Private Sub NotifyIcon1_DoubleClick(ByVal sender As
          Object, ByVal e As System.EventArgs) Handles
            NotifyIcon1.DoubleClick, MenuItemShow.Click
Me.ShowInTaskbar = True
Me.WindowState = FormWindowState.Normal
Me.NotifyIcon1.Visible = False
Me.NotifyIcon1.Icon = New Icon(Me.GetType,
                                        "Normal.ICO")
End Sub
```

Come si vede, si intercetta l'evento *LocationChanged* della form e si verifica se la form è stata mini-

mizzata. A quel punto, se l'utente preferisce la minimizzazione nella *Notification Area*, si toglie la form dalla TaskBar e si visualizza l'iconcina. In caso di doppio click sull'iconcina, o in caso di click sul menu contestuale associato all'iconcina alla voce *Show*, si provvede a ripristinare la form, e a ripristinare l'icona (che potrebbe essere stata cambiata per segnalare all'utente l'arrivo di uno o più eventi).

SELEZIONE DI FILE E CARTELLE

La selezione viene effettuata tramite una form custom, che ci permette di selezionare il file o la cartella da monitorare. In caso di cartella, si può impostare un filtro sui file da cercare, ed è possibile selezionare gli eventi da controllare (Fig. 6). Per ottenere questa form sono stati usati il controllo *TaskPane*, dove nei singoli *Pane* sono stati inseriti un controllo *BlendPanel* per dare lo sfondo e i controlli standard .NET richiesti (*TextBox*, e *CheckBox*), mentre per cartelle e file si sono utilizzati il *FolderViewer* e il FileViewer del VB Power Pack. La *TaskPane* è completamente customizzabile, si possono scegliere i colori dello sfondo, del testo e si possono impostare le sfumature per le caselle dei titoli che servono anche a far "collassare" i vari *Pane*.
All'interno del singolo *Pane* può essere messo qualsiasi tipo di controllo. La parte più interessante della form è quella che riguarda la sincronizzazione tra i due *Viewer:*

```
Private Sub FolderViewer1_NodeClicked(ByVal sender
                         As System.Object, ByVal e As
                    VbPowerPack.Folder ViewerEventArgs)
                       Handles FolderViewer1.NodeClicked
Try
Me.FileViewer1.Path = Me.FolderViewer1.CurrentFolder
Catch ex As Exception
Me.FileViewer1.Path = "My Computer"
End Try
File = ""
End Sub
Private Sub FileViewer1_ItemDoubleClicked(ByVal sender
             As Object, ByVal fvea As VbPowerPack. FileViewer
             EventArgs) Handles FileViewer1.ItemDoubleClicked
If fvea.Item.IsFolder Then
Me.FolderViewer1.CurrentFolder = fvea.Item.FullPath
Else
DialogResult = DialogResult.OK
End If
End Sub
```

Se viene selezionato un nodo nel *FolderViewer*, viene settato il Path nel *FileViewer.* In questo caso ci può essere un problema in caso della cartella *My Computer,* che va settata esplicitamente in caso di eccezione. Se viene selezionato un elemento del *FileViewer,* viene controllato se è un file o una cartella. Nel primo caso si esce, nel secondo si imposta il *FullPath* del *FolderViewer* alla cartella selezionata.
Per il resto, la form contiene solo la definizione delle proprietà che verranno lette dalla routine che crea un nuovo *FileSystemWatcher.* L'unica altra parte di rilievo è la gestione dei *Flag* per decidere gli eventi del *FileSystem* da monitorare. Per questo viene definita una proprietà che mappa il valore dei vari *CheckBox* (se ne mostrano solo alcuni):

```
Public ReadOnly Property NotifyFilt() As Integer
Get
Dim val As Integer = 0
If Me.chkAttributes.Checked Then val +=
                              NotifyFilters.Attributes
If Me.chkCreatTime.Checked Then val +=
                           NotifyFilters.CreationTime
Return val
End Get
End Property
```

FUNZIONALITÀ ACCESSORIE

Per rimuovere un file o una cartella dal monitoraggio, si utilizza il seguente codice, associato alla voce di menu e al pulsante *Delete* della Toolbar.

```
Private Sub UtTMain_DeletePressed(ByVal sender As
 System.Object, ByVal e As System.EventArgs) Handles
          UtTMain.DeletePressed, MenuItemRemove.Click
If Me.TabCMain.SelectedIndex > -1 Then
RemoveHandler CType(Me.Watchers(
            Me.TabCMain.SelectedIndex), FileWatcher)
      .Watcher.Changed, AddressOf Me.OnDiskChanged
Me.Watchers.RemoveAt(Me.TabCMain.SelectedIndex)
Me.TabCMain.TabPages.RemoveAt(Me.TabCMain.SelectedIndex)
Me.StatusBar1.Text = "Currently monitoring " +
           Watchers.Count.ToString + " file(s)/dir(s)..."
End If
End Sub
```

Praticamente si controlla se c'è una TabPage selezionata e si rimuove il corrispondente elemento dell'ArrayList, ricordandosi di deregistrare l'evento dal *FileSystemWatcher.* Per muoversi avanti e indietro si incrementa e decrementa l'indice della pagina attiva del *TabCMain*, mentre per ripulire il TextBox, si usa il seguente codice:

```
CType(Me.TabCMain.TabPages(Me.TabCMain.SelectedIndex)
                       .Controls(0), TextBox).Text = ""
```

I tooltip dei bottoni della Toolbar non sono adatti alla nostra applicazione, e quindi all'atto del *Form_*

MULTITHREADING E WINDOWS FORMS

Per un approfondimento sul problema del multithreading e l'uso delle Windows Forms si può guardare il seguente articolo di Corrado Cavalli (richiede registrazione al sito):

www.ugidotnet.org/articles/articles_read.aspx?ID=19

***Fig. 7:* Form per modificare le opzioni**

Il sito ufficiale del VB Power Pack è www.gotdotnet.com/Workspaces/Workspace.aspx?id=167542e0-e435-4585-ae 4f-c111fe60ed58 **da cui si possono scaricare i sorgenti, gli installer sia per Visual Studio .NET 2003, sia per la 2002. C'è anche una message board e un sistema per il reporting dei problemi.**

Load li cambiamo, utilizzando la seguente istruzione (ne viene mostrato solo uno):

```
Me.SearchUtilityButton1.ToolTipText = "New"
```

Le informazioni di configurazione vengono lette nel *Form_Load*, e se è richiesta la partenza minimizzata, viene attivata:

NOTA

CONTROLLI UTILIZZATI NEL PROGETTO

Nel progetto si sono utilizzati tutti i controlli presenti nel VB PowerPack. Nella *FormMain* ci sono la *NotificationWindow* e l'*UtilityToolbar*, nella *FormAbout* ci sono il *BlendPanel* e l'*ImageButton*, mentre nella *FormChoose* ci sono la *TaskPane* (che contiene due BlendPanel), il *FolderViewer* e il *FileViewer*.

```
Me.ShowToast = Boolean.Parse(Configuration.
ConfigurationSettings.AppSettings("ShowToast").ToString())
Me.StartMinimized = Boolean.Parse(Configuration.
    ConfigurationSettings.AppSettings("StartMinimized")
                                            .ToString())
Me.MinimizeInNotifyArea = Boolean.Parse(Configuration.
  ConfigurationSettings.AppSettings("MinimizeInNotifyAr
                                        ea").ToString())
If Me.StartMinimized Then
Me.WindowState = FormWindowState.Minimized
End If
```

Per modificare le preference si utilizza una form apposta (Fig. 7).

Fig. 8: Form di About

Per salvarle si deve operare direttamente sul file XML contenente la configurazione (viene mostrata la modifica di un solo campo):

```
Dim xmlDom As New Xml.XmlDocument
Dim xmlElm As Xml.XmlElement
xmlDom.Load(Application.ExecutablePath + ".config")
xmlElm = CType(xmlDom.SelectSingleNode(
                      "//add[@key='StartMinimized']"),
                                       Xml.XmlElement)
xmlElm.SetAttribute("value", Me.StartMinimized.ToString)
xmlDom.Save(Application.ExecutablePath + ".config")
```

Un effetto interessante può essere ottenuto nella form di About del programma, utilizzando il *BlendPanel*, una PictureBox standard .NET e un controllo *ImageButton*, come mostrato in Fig. 8. L'unica accortezza è che sembra che la proprietà *DialogResult* dell'*ImageButton* non funziona, quindi ho intercettato l'evento *Click* e l'ho impostata a mano. È stata implementata anche la possibilità di rivedere (quando il programma è minimizzato nella *Notification Area*) l'ultimo messaggio registrato, per non dover per forza aprire l'interfaccia utente. Si noti che il messaggio mostrato è l'ultimo arrivato al programma, non l'ultimo mostrato (potrebbero essere diversi se arrivano tanti messaggi contemporaneamente).

PROBLEMI RISCONTRATI

Il VB Power Pack è un progetto giovane, che non è esente da qualche piccolo problema. Tra i più fastidiosi si registra nella gestione della *NotificationWindow*, che appare (sul mio sistema) in basso a sinistra invece che in basso a destra come il *Messenger*. Inoltre, se si seleziona la modalità *Slide*, la finestrella viene srotolata nella direzione sbagliata (secondo me, ma la spiegazione presente sul sito ufficiale recita che è così *"by design"*…). Questi problemi possono essere sistemati intervenendo sui sorgenti. Anche se, pensandoci bene, è più che altro una questione di gusti! Ci sono altri piccoli problemi come il *DialogResult* dell'*ImageButton* che non funziona a dovere, o il Path del *FileViewer* che non accetta la selezione di *"My Computer"* fatta dal *FolderViewer*, ma sono problemi semplicissimi da risolvere. La disponibilita sul Workspace *GotDotNet* dei sorgenti ci è poi d'aiuto nel caso volessimo customizzare i controlli a nostro piacere.

CONCLUSIONI

Abbiamo visto come realizzare un'applicazione molto semplice, ma dotata di tutte le feature delle applicazioni più moderne (minimizzazione nella *Notification Area*, finestrella toast per la notifica dei messaggi, *TaskPane* alla Windows XP, dialog di selezione dei file customizzata, etc…). Nel prossimo numero estenderemo l'applicazione introducendo l'uso di alcuni Application Block disponibili gratuitamente sul sito MSDN, che ci permetteranno di aggiungere altre feature interessanti alla nostra applicazione.

Lorenzo Barbieri

L'AUTORE

Lorenzo Barbieri è laureato in Ingegneria Informatica e lavora come *Trainer* e *Consulente* sulle tecnologie Microsoft. È *Microsoft Certified Trainer* ed è certificato *MCSD.NET* e *MCDBA* su SQL Server 2000. Può essere contattato attraverso i suoi blog:

http://weblogs.asp.net/lbarbieri e http://blogs.ugidotnet.org/lbarbieri

Fig. 9: NotificationWindows in basso a destra dello schermo

Un'introduzione al nuovo IDE open-source

Sviluppare in Java con Eclipse 3

parte seconda

Continuiamo la nostra esplorazione del mondo di Eclipse con la gestione degli eventi nel Visual Editor e un'introduzione al framework grafico SWT, ideale per prestazioni e flessibilità

Visto l'enorme successo del progetto open-source Eclipse presso la comunità Java e oltre, abbiamo iniziato nel numero scorso una mini-serie di presentazione di Eclipse 3.0, l'ultima versione del popolare tool di sviluppo rilasciata a luglio di quest'anno. Volendo dare una panoramica delle capacità dell'ambiente di sviluppo in relazione ad altri prodotti similari, abbiamo anche introdotto ai lettori il sottoprogetto VEP (*Visual Editor Project*), un plug-in aggiuntivo che dà la possibilità di creare interfacce grafiche in maniera visuale e semplice. Siamo partiti dalla creazione di una applicazione calcolatrice, la cui interfaccia è stata appunto disegnata con VE ed è rappresentata nella Fig. 1.
Abbiamo visto come sia possibile gestire il layout *GridBagLayout* di Swing per collocare sulla finestra i vari componenti e dare loro una posizione e dimensione. Oggi partiamo invece dalla gestione degli eventi.

Fig. 1: *L'interfaccia grafica della semplice calcolatrice Java d'esempio*

CREARE E GESTIRE GLI EVENTI CON VE

Grazie al plug-in VE possiamo aggiungere facilmente degli ascoltatori ai vari elementi che un form contiene. Cliccando con il tasto destro su un componente, si aprirà il menu contestuale, nel quale troviamo la voce *Events* (vedi Fig. 2). Da lì è possibile aprire la finestra di dialogo *Add Events* (vedi Fig. 3), nella quale si può scegliere tra tutte le tipologie di ascoltatore che il componente selezionato supporta, oppure utilizzare le scorciatoie per creare direttamente un ascoltatore per gli eventi più comuni del tal componente.

Fig. 2: *Il menu per l'aggiunta di gestori d'evento ai componenti di una finestra*

Il codice di evento che viene generato prevede la creazione di una classe anonima passata come parametro del metodo di registrazione dell'evento, come nell'esempio che segue:

Fig. 3: *La finestra con tutti i gestori d'evento disponibili per un componente*

```
button.addActionListener(new java.awt.event.
                                          ActionListener() {
  public void actionPerformed(
                        java.awt.event.ActionEvent e) {
    // TODO Auto-generated Event stub actionPerformed()
  }
});
```

Non è quindi possibile, utilizzando l'aiuto fornito dal VE, creare una sola classe registrata come ascoltatrice per più di un componente. I pulsanti Swing, però hanno anche una proprietà *azione* con cui si può specificare una classe da usare per gestire l'evento *actionPerformed*. Tale classe deve essere una derivata di *javax.swing.AbstractAction* e potrà essere associata alla proprietà *Action* dei vari componenti di una finestra tramite la view *"Properties"*. Nella nostra calcolatrice abbiamo così usato una sola classe *(CalcBtnAction)* per gestire il click di tutti i pulsanti presenti: ognuno contribuirà una cifra o una operazione a seconda del testo che visualizza.

Fig. 4: **Il menu di selezione per la visualizzazione degli eventi della Java Beans view**

Nel momento in cui abbiamo degli eventi gestiti nel nostro codice, la view *"Java Beans"* diviene un prezioso strumento di consultazione: essa, come abbiamo visto nel numero scorso, offre una visualizzazione della struttura dei componenti presenti in una classe visuale, ordinati gerarchicamente e selezionabili in modo da passare direttamente al codice nell'editor o all'elemento visuale nel designer grafico. Oggi, invece, scopriamo che, tramite un menu a comparsa (quello della Fig. 4), possiamo anche vedere in quella struttura tutti gli eventi che sono stati associati ai vari controlli ed eventualmente eliminare quelli che non servono più. Il menu offre tre opzioni: la prima (*No Events*) fa sì che non vengano mostrati gli eventi nella view, mentre *Show Events*

Fig. 5: **la Java Beans view mostra gli eventi dei vari elementi**

mostra quali eventi sono gestiti per quali componenti (come in Fig. 5). La terza opzione (*Expert Events*) offre anche il dettaglio sul modo in cui il gestore è costruito (implementazione anonima di una interfaccia o estensione anonima di una classe), così che sapete esattamente che tipologie di gestori sono in esecuzione nella vostra applicazione (Fig. 6). Notate che, tramite la finestra di aggiunta degli eventi, (quella di figura *wineventi*) possiamo utilizzare una stessa classe anonima per gestire eventualmente più eventi se essa lo consente: sostanzialmente se un componente ha già una classe di ascolto la cui interfaccia offre altri metodi di gestione eventi, allora la dialog box di aggiunta eventi darà la possibilità di riutilizzare quell'ascoltatore esistente abilitando il radio button relativo.

Fig. 6: **La modalità avanzata di visualizzazione eventi della Java Beans view**

LO STANDARD WIDGET TOOLKIT

Iniziamo adesso a parlare della nostra seconda applicazione di esempio, quella con cui andremo a generare un file PDF tramite una interessante libreria di oggetti Java completamente gratuita e open-source. Ma, prima di parlare di PDF, vorrei introdurre una nuova tecnologia di sviluppo di interfacce a finestre che si è affacciata al mondo con l'uscita di Eclipse. Si tratta di SWT (lo *Standard Widget Toolkit*), una libreria grafica al pari di AWT e Swing, nata in casa IBM e adesso parte integrante del progetto Eclipse, rispetto al quale comunque si propone come API indipendente. L'esigenza che ha portato a SWT è quella di unire alla portabilità del codice sviluppato anche una buona performance nell'esecuzione delle applicazioni. La prima tecnologia visuale di Java fu l'AWT (*Abstract Window Toolkit*). Esso è molto ef-

NOTA

LO STANDARD WIDGET TOOLKIT

SWT è stato sviluppato da IBM come base di appoggio per il progetto Eclipse con l'obiettivo di supplire alle carenze dei due sistemi grafici Java in esistenza (AWT e Swing). Per una guida all'architettura interna di questa nuova API potete fare riferimento a due interessanti articoli sul sito di Eclipse che presentano le caratteristiche di questo nuovo framework per creare applicazioni a finestre in Java:
www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html
www.eclipse.org/articles/swt-design-2/swt-design-2.html

ficiente in quanto è basato sull'esecuzione di codice nativo, ma ha sacrificato alla portabilità il range di componenti offerti: affinché ogni controllo grafico potesse essere creato nativamente sul sistema operativo di esecuzione dell'applicazione, l'offerta disponibile è limitata ai soli controlli disponibili su tutti gli ambienti di destinazione di Java. AWT quindi risulta molto povero di scelta e poco flessibile. Swing invece, che arriva in forma finale con Java 1.2, rappresenta un cambio di politica radicale: non si usano più chiamate al sistema operativo sottostante per creare i controlli e gli elementi grafici delle finestre, tutto viene fatto al 100% in Java. Questa alternativa risulta quindi sicuramente portabile e ricca di controlli e gadget di ogni genere, perché adesso non ci si deve più soffermare a valutare quali sistemi supportino quali elementi. Lo svantaggio di Swing però risiede nelle scarse performance. Rendere tutto in Java costa molto in termini di risorse e tempo e, nonostante i grandi miglioramenti fatti con le varie implementazioni dell'API, la sua performance, pur accettabile, non è equiparabile a tecnologie parallele. Entra così in scena SWT: l'idea è questa volta di creare una libreria Java comune alle varie piattaforme, la quale a sua volta si appoggia su una libreria interna di chiamate native. L'implementazione di questa libreria nativa varia a seconda del sistema operativo facendo uso di oggetti specifici che inoltrano chiamate via JNI a librerie dinamiche del sistema operativo (.dll, .so, etc.) sviluppate in C appositamente per ogni piattaforma supportata. A differenza di AWT, però, questo approccio non si limita ad uno scarno set comune di componenti: dove i sistemi operativi non offrono supporto, la libreria nativa deve supplire con implementazioni 100% Java. In questo modo si ottengono applicazioni efficienti, portabili e flessibili e la stratificazione a tre livelli consente comunque di offrire ai programmatori un API universale. Eclipse ovviamente è la prima applicazione Java ad essere scritta con questa tecnologia.

Fig. 7: ***La finestra di impostazione delle librerie di sviluppo***

SWT LAYOUTS

I layout in SWT come in Swing determinano la posizione e la dimensione degli elementi all'interno di un contenitore. Un'ottima introduzione ai vari layout disponibili in SWT (sensibilmente diversi da quelli che si trovano in Swing) la trovate sul sito di Eclipse all'indirizzo www.eclipse.org/articles/Understanding%20Layouts/Understanding%20Layouts.htm

MANO AL CODICE!

Scriviamo adesso anche noi la nostra applicazione SWT, cominciando col creare un nuovo progetto Java: io l'ho chiamato *"PdfGeneratorWithEclipse"*, ma qualunque nome di vostra scelta andrà bene. Selezioniamo poi le proprietà del progetto per aggiungere le librerie necessarie allo sviluppo con SWT: *right-click* sul progetto nella vista *"Package Explorer"* o *"Navigator"*, menu *Properties*, item *Java Build Path*, pagina *Libraries*, come in Fig. 7. Da lì cliccate sul pulsante *Add Library…* e selezionate per l'appunto *Standard Widget Toolkit* (SWT), ignorando per adesso l'opzione relativa al supporto per JFace (Fig. 8). In questo modo avremo a disposizione tutti i package e le classi necessarie per sviluppare applicazioni SWT. Creiamo dunque la nostra classe principale, che offrirà anche la finestra iniziale con le opzioni di base. Dal menu *File*, o cliccando sempre col destro sul progetto, selezioniamo *New->Visual Class* (se necessario passate per *Other…* per avere la lista completa degli elementi disponibili: ricordate che i menu di Eclipse sono contestuali e dipendono quindi dalla prospettiva). Configurate così la nuova classe: nome del package *"fedmest.eclipse.pdfgen"*, nome della classe *"MainWin"*, lo stile *"SWT->Application"* e selezionate la spunta di creazione del metodo *main*. Il metodo *main* generato contiene già il codice necessario alla creazione della finestra principale, la cui sostanza (seppur con variazione sul tema) è la seguente:

Fig. 8: ***Selezioniamo la libreria SWT per il nostro nuovo progetto***

```
public static void main(String [] args) {
  Display display = new Display();
  Shell shell = new Shell(display);
  shell.open();
  while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
      display.sleep(); }
  display.dispose();
}
```

Il *Display* rappresenta una sessione di lavoro SWT ed è responsabile per la comunicazione con il sistema operativo sottostante. Una delle sue funzioni più importanti è quella di riprodurre il ciclo degli eventi della piattaforma di esecuzione e riportarlo all'interno del codice Java (il blocco *while* del metodo *main* qui sopra). Tipicamente, ogni applicazione ha biso-

gno di un solo *Display*. La *Shell* invece è una finestra aperta direttamente sul desktop del sistema operativo: è una finestra principale se il parent è il *Display*, viene detta invece secondaria se il parent è un'altra *Shell*.
Passiamo adesso alla creazione visuale del nostro form. Non ci sono differenze enormi rispetto a quanto detto per *Swing*, cambiano però i layout a disposizione: SWT offre un layout *null* (basato su posizionamento e dimensionamento statici), un *FillLayout* (gli elementi hanno la stessa dimensione ed occupano tutto lo spazio libero disposti in una sola riga o colonna), un *RowLayout* (elementi disposti per righe o colonne con varie opzioni), un *GridLayout* (elementi disposti in una griglia) e un *FormLayout* (il più flessibile, ma senza supporto visuale in VEP per adesso). Io ho scelto il *GridLayout*, che seleziono per la proprietà *Layout* nella view *"Properties"*: è richiesto di preimpostare il numero di colonne (a differenza del *GridBagLayout* di Swing), cosa che possiamo fare nella stessa *view*, o nella finestra *Customize Layout* (Fig. 9), raggiungibile tramite l'omonimo menu contestuale sul form. La vostra finestra completa dovrebbe assumere una forma simile a quella di figura form.

Fig. 9: ***Dialog di configurazione del layout e dei dati di layout di ogni componente***

Fig. 10: ***Il nostro form iniziale per l'applicazione di generazione di PDF***

Aggiungiamo ancora gli eventi *widgetSelected* (il click per farla breve) ai tre pulsanti che abbiamo messo sul form: per adesso, l'unico evento che implementeremo è quello del tasto di scelta file, che avrà il codice seguente, nel quale, come potete immaginare, *FileDialog* è una classe SWT che richiama la dialog box di selezione file del sistema operativo:

```
public void widgetSelected(
        org.eclipse.swt.events.SelectionEvent e) {
  FileDialog dlg = new FileDialog(sShell);
  String filename = dlg.open();
```

```
  if ( filename != null )
    text.setText(filename); }
```

Adesso siamo pronti per vedere il nostro form in azione. Per eseguire il codice Java di un'applicazione SWT, però, abbiamo ancora bisogno di configurare il percorso della libreria nativa specifica di Windows. Se ricordate, infatti, abbiamo detto che uno strato di SWT fa chiamate ad una *.dll* (*.so* per Linux e così via) scritta appositamente per il sistema operativo di destinazione: ecco allora che dobbiamo passare alla JVM il percorso di tale libreria dinamica affinché SWT funzioni. Aprite il dialog del menu *Run...* e aggiungete questo valore alla casella di testo *"VM Arguments"* di una nuova configurazione per l'esecuzione di applicazioni Java

Fig. 11: ***Impostazione del percorso della libreria nativa SWT per la JVM di test di Eclipse***

```
-Djava.library.path="D:\eclipse\plugins\
        org.eclipse.swt.win32_3.0.0\os\win32\x86"
```

sostituendo il percorso *D:\eclipse* con quello effettivo di installazione di Eclipse sulla vostra macchina (vedi Fig. 11). Adesso la vostra applicazione è pronta per essere eseguita!

CONCLUSIONI

Per oggi terminiamo qui il nostro lavoro. Abbiamo già messo molta carne al fuoco, se tenete conto del fatto che SWT è un intero sistema di gestione di interfacce per Java.
Approfonditene lo studio perché ne vale la pena: Eclipse è un esempio di quanto efficace, performante e "visually appealing" questa nuova API possa essere. Nella prossima parte porteremo avanti la nostra applicazione di generazione di PDF: parleremo di *FormLayout*, decisamente il più flessibile ed interessante tra quelli proposti da SWT, e predisporremo un form per chiedere all'utente cosa vuole vedere sul file PDF generato.

Federico Mestrone

CONTATTA L'AUTORE

DOMANDE?

Se avete bisogno di qualche aiuto, potete scrivermi a federico.mestrone@ioprogrammo.it
Anche se la risposta può farsi attendere molto, soprattutto quando sono sotto stress, normalmente arriva sempre!

NOTA

APRIRE I PROGETTI ALLEGATI

In allegato all'articolo trovate tutto il workspace utilizzato per creare le applicazioni di esempio. Potete aprirlo estraendo il file *.zip* e selezionando la cartella *workspace* risultante come nuovo ambiente di lavoro tramite il menu *File->Switch Workspace...*. Alternativamente potete importare i singoli progetti che si trovano sotto la cartella indicata (in questo caso *JavaDevWithEclipse* o *PdfGeneratorWithEclipse*) tramite il menu Import... con origine dati *Existing Project into Workspace*.

Il namespace System.IO

File: input e output

In questo appuntamento descriveremo le classi messe a disposizione da VB.NET che permettono di leggere e scrivere su file: indispensabile per qualsiasi progetto

La classe *Stream* è una classe astratta che permette di gestire una generica sequenza di byte *(flusso)* nelle operazioni di input/output. Uno *Stream* è un flusso di byte che può essere, di volta in volta, identificato con un oggetto diverso: un file, una periferica di input/output, un canale di comunicazione tra processi, o ancora un socket TCP/IP. L'utilizzo della classe *Stream* fornisce al programmatore un mezzo per gestire la lettura e l'invio di dati tra il programma e la periferica, senza che questi si debba preoccupare di come sia realizzato il collegamento tra dati e periferiche. Esistono stream per le diverse entità di memorizzazione (stream per file, stream di memoria, stream di rete, ecc.).
Sugli stream è possibile eseguire tre operazioni fondamentali: *lettura, scrittura* e *ricerca.*

- La *lettura* consiste nel trasferire informazioni da un generico flusso ad una struttura di dati, ad esempio una matrice di byte (come vedremo in seguito).
- Con la *scrittura* possiamo trasferire informazioni da una struttura di dati ad un flusso.
- La *ricerca* consiste nella richiesta, ed eventuale modifica, della posizione corrente di uno specifico gruppo di byte all'interno di un flusso.

Non tutti i tipi di stream, tuttavia, offrono supporto a tutte e tre le operazioni fondamentali. Per verificare le operazioni permesse, è possibile utilizzare le proprietà:

- **CanRead** restituisce *True* se il flusso supporta la lettura.
- **CanWrite** restituisce *True* se il flusso supporta la scrittura.
- **CanSeek** restituisce *True* se il flusso supporta la ricerca.

Altre proprietà e metodi interessanti sono:

- **Length** restituisce la lunghezza dello *Stream* in byte.
- **Position** restituisce la posizione corrente all'interno dello stream (Il punto di partenza delle operazioni di lettura o scrittura).
- Il metodo **Seek** permette di riposizionare il puntatore all'interno dello stream, è quindi possibile muoversi avanti e indietro lungo uno stream per stabilire il nuovo punto di partenza delle successive operazioni di lettura o scrittura.
- Il metodo **SetLength** permette di modificare la lunghezza dello *Stream* rendendolo pari al valore passato come argomento. Se il valore indicato è minore della lunghezza dello stream, allora lo stream viene troncato. Se il valore indicato è maggiore della lunghezza dello *stream*, allora lo stream viene esteso.
- Il metodo **Read** legge una sequenza di byte, della lunghezza indicata, a partire dalla posizione iniziale specificata come argomento. Ha come effetto di far avanzare il puntatore allo *Stream*. Si può utilizzare anche il metodo *ReadByte* che, però, legge e restituisce un singolo byte.
- Il metodo **Write** scrive un numero di byte, della lunghezza indicata, a partire dalla posizione iniziale specificata come argomento. Anche questo metodo fa avanzare il puntatore allo *Stream*. Si può utilizzare anche il metodo *WriteByte* che, però, scrive un unico byte.
- Il metodo **Close** chiude lo stream e libera tutte le risorse di sistema associate. Se si compie una operazione su un flusso chiuso, in seguito ad una chiamata a *Close,* VB.Net potrebbe generare un'eccezione. Se il flusso è già chiuso, una successiva chiamata al metodo *Close,* non genera eccezioni.
- Il metodo **Flush** permette di svuotare uno *Stream* memorizzato in un buffer ed assicura che tutto il relativo contenuto sia scritto nel corrispondente dispositivo fisico. Il metodo *Flush* non ha alcun effetto sugli stream non bufferizzati.

Come accennato in precedenza, la classe *Stream* è una classe astratta, pertanto non è possibile creare direttamente un oggetto *Stream* e, di conseguenza, vi troverete raramente ad utilizzare una variabile *Stream* all'interno del codice, mentre utilizzerete più spesso classi, che derivano da *Stream.*

Nel modulo *ControlChars* sono disponibili costanti utilizzate come caratteri di controllo che possono essere inserite in qualsiasi punto del codice. In particolare la costante *ControlChars.CrLf* rappresenta la combinazione di ritorno a capo e avanzamento riga *(Chr(13) + Chr(10)).*

Le classi che derivano da *Stream*, implementano ulteriori metodi e proprietà, peculiari dell'entità di memorizzazione che rappresentano. Ad esempio, la classe *FileStream*, che permette di accedere ai file (comprese le periferiche standard di input, output ed errore), espone la proprietà *Handle* che restituisce l'handle del file di sistema operativo (gli handle sono dei descrittori numerici, ciascuno associato ad un file aperto, che il programma utilizza per effettuare le operazioni di scrittura, lettura e posizionamento) ed i metodi *Lock* e *Unlock* che permettono di bloccare o sbloccare una parte del file. La classe *FileStream* utilizza, inoltre, le enumerazioni *FileAccess, FileMode* e *FileShare* che rappresentano i flag utilizzati dai costruttori della classe *FileStream* e determinano la modalità con cui viene creato, aperto e condiviso un file.

ACCEDERE A DATI STRUTTURATI

Abbiamo visto come il generico oggetto *Stream* sia in grado di leggere e scrivere byte singoli o gruppi di byte, ma, nei casi reali, si rende necessario poter leggere e scrivere dati strutturati, come una riga di testo o un numero. Per questo scopo, VB.Net mette a disposizione degli oggetti ausiliari più specializzati, in particolare:

- Le classi **BinaryReader** e **BinaryWriter** permettono di leggere e scrivere dati primitivi, come un numero intero o una stringa codificata, in formato binario.
- Le classi **StreamReader** e **StreamWriter** permettono di leggere e scrivere stringhe di testo in formato ANSI, come quelle che sono lette o scritte in un file di testo. A differenza di *BinaryReader* e *BinaryWriter,* le informazioni sono trattate come testo piuttosto che come dati binari.
- **TextReader** e **TextWriter** sono classi astratte che definiscono le modalità di utilizzo delle stringhe di testo in formato Unicode. Le classi *StringReader* e *StringWriter* derivano da *TextReader* e *TextWriter.*

Riassumendo: la classe *Stream* è stata progettata per l'input e l'output dei byte, *TextReader* e *TextWriter* sono stati progettati per l'input e l'output di caratteri, *BinaryReader* e *BinaryWriter* sono stati progettati per la lettura e scrittura dei tipi primitivi in modalità binaria (anziché testo).

LEGGERE DA UN FILE DI TESTO

Per leggere da un file di testo si può utilizzare un oggetto *StreamReader*. La classe *StreamReader* utilizza una codifica per convertire le matrici di byte in stringhe di caratteri. Come impostazione predefinita viene utilizzata la codifica *UTF-8* (per gli altri tipi di codifica, vi rimando al box a lato). Questa classe espone dieci costruttori in overload, alcuni dei quali consentono di rilevare automaticamente la codifica di un file oppure di specificarla come parametro. Esistono, pertanto, diversi modi per ottenere un riferimento ad un oggetto di questo tipo, ad esempio:

- Si può creare un oggetto *FileStream* con il metodo *Open* della classe *File* (descritta nell'articolo precedente). Ricordiamo che il metodo *Open* apre un oggetto *FileStream,* nel percorso specificato e con le modalità di accesso passate come argomento. Dopo aver creato l'oggetto di tipo *FileStream* si deve passarlo, come argomento, al metodo costruttore dello *StreamReader*

```
Dim OggettoStream As FileStream = File.Open(
                              "C:\MioFile.txt", FileMode.Open,
                    FileAccess.ReadWrite, FileShare.ReadWrite)
Dim LeggiStream As New StreamReader(OggettoStream)
```

- Si può creare un oggetto *FileStream* utilizzando il costruttore e, quindi, passarlo al metodo costruttore dello *StreamReader:*

```
Dim OggettoStream As New FileStream(
                              "C:\MioFile.txt", FileMode.Open)
Dim LeggiStream As New StreamReader(OggettoStream)
```

- Si può passare al costruttore il percorso completo del file da leggere:

```
Dim LeggiStream As New StreamReader("c:\MioFile.txt")
```

- Si può passare al costruttore il percorso completo del file e la codifica dei caratteri da utilizzare:

```
Dim LeggiStream As New StreamReader(
          "c:\MioFile.txt", System.Text.Encoding.ASCII)
```

- Si può passare al costruttore il percorso completo del file, lasciando al sistema il compito di rilevare la codifica:

```
Dim LeggiStream As New StreamReader(
                                    "c:\MioFile.txt", True)
```

Dopo aver ottenuto un riferimento all'oggetto *StreamReader*, è possibile utilizzare uno dei metodi messi a disposizione, che permettono di leggere uno o più caratteri per volta oppure un'intera riga di testo.

- Il metodo **Peek** restituisce il codice del successivo carattere disponibile nello stream senza utilizzar-

Per mantenere il codice il più compatto possibile, si può utilizzare l'istruzione *Imports* che semplifica l'accesso alle classi, eliminando la dichiarazione in modo esplicito il nome completo del namespace che le contiene. Le istruzioni *Imports* devono sempre essere scritte nella parte superiore del file nel quale volete utilizzarle, prima di qualunque altro codice. In tutti gli esempi di codice, si dà per scontato l'inserimento della seguente istruzione *Imports:*

```
Imports System.IO
```

lo effettivamente. Nel caso in cui non esistano più caratteri nello stream, *Peek* restituisce il valore speciale *-1*.
Questo metodo viene utilizzato come test sul valore *-1* per verificare la condizione di *end-of-file*.

- Il metodo **ReadLine** consente di leggere un'intera riga di caratteri dal flusso in esame.
- Il metodo **Read** consente di leggere un carattere per volta. Questo metodo, legge il carattere successivo e fa avanzare di un carattere la posizione corrente.
- Il metodo **ReadToEnd** permette di leggere tutti i caratteri rimanenti, dalla posizione corrente fino alla fine del flusso. Se la posizione corrente corrisponde con la fine del flusso, restituisce una stringa vuota *("")*.
- Il metodo **Close** chiude l'oggetto *StreamReader* e libera tutte le risorse di sistema.

Fig. 1: *Il risultato della nostra applicazione di esempio*

NOTA

Una riga è definita come una sequenza di caratteri seguita da un avanzamento riga ("\n") o da un ritorno a capo immediatamente seguito da un avanzamento riga ("\r\n").

La maggior parte degli oggetti *stream* memorizza i dati in un buffer. Ad esempio, quando si scrive in uno *stream* di file, i dati non vengono scritti subito su disco ma vengono memorizzati in un buffer ed inviati al file alla chiusura dello stream oppure quando si invoca il metodo *Flush*.

A questo punto abbiamo tutti i mezzi per scrivere una applicazione che legga tutte le righe di testo presenti in un file, e le mostri in una finestra. Dopo aver creato una nuova applicazione Windows, nella finestra di progettazione dovremo inserire:

- Una casella di testo *(TextBoxPathFile)* in cui l'utente dovrà scrivere il percorso completo del file da leggere.
- Una casella di testo *(TextBoxRisultato) multiRiga* (con la proprietà *MultiLine* pari a *True*), in cui sarà mostrato il contenuto del file.
- Un pulsante *(ButtonLeggi)* che avvia la lettura e valorizza la casella di testo con il risultato di lettura.

Il codice necessario dovrà essere scritto, ovviamente, nell'evento *ButtonLeggi_Click:*

```
Private Sub ButtonLeggi_Click(ByVal sender
        As System.Object, ByVal e As System.EventArgs)
                                  Handles ButtonLeggi.Click
 'Apertura del file e creazione dell'oggetto StreamReader
   Dim OggettoStream As FileStream = File.Open(
                    TextBoxPathFile.Text, FileMode.Open,
                FileAccess.ReadWrite, FileShare.ReadWrite)
   Dim LeggiStream As New StreamReader(
                                         OggettoStream)
   'dichiarazione della variabile che memorizza
                                             il testo letto
   Dim Contenuto As String
   'ciclo fino alla fine del file
   While LeggiStream.Peek <> -1
     'concate na le righe di testo aggiungendo il
                                      carattere di fine riga
     Contenuto = Contenuto +
         LeggiStream.ReadLine + ControlChars.CrLf
   End While
   'Ricordatevi di Chiudere sempre uno StreamReader
   LeggiStream.Close()
   'visualizza il risultato nel TextBox
   TextBoxRisultato.Text = Contenuto
End Sub
```

Nel caso in cui il file non esista, viene generata un'eccezione del tipo: *System.IO.FileNotFoundException*. Se, invece di leggere una riga per volta, vogliamo leggere l'intero contenuto del file in un colpo solo, possiamo sostituire il ciclo *While*, ed il codice al suo interno, con l'istruzione:

```
Contenuto = LeggiStream.ReadToEnd
```

SCRIVERE IN UN FILE DI TESTO

Per scrivere in un file di testo si può utilizzare l'oggetto *StreamWriter*. Come nel caso di *StreamReader*, esistono molti modi per creare un oggetto *StreamWriter:*

- Si può creare un oggetto *FileStream* con il metodo *Open* della classe *File* e passarlo al metodo costruttore dello *StreamWriter:*

```
 Dim OggettoStream As FileStream = File.Open(
                    "C:\Nuovofile.txt", FileMode.Create,
                 FileAccess.ReadWrite, FileShare.None)
    Dim ScriviStream As New StreamWriter(
                                         OggettoStream)
```

- Si può creare un oggetto *FileStream* utilizzando il costruttore e, quindi, passarlo al metodo costruttore dello *StreamWriter:*

```
Dim OggettoStream As New FileStream(
                      "C:\Nuovofile.txt", FileMode.Open)
Dim ScriviStream As New StreamWriter(OggettoStream)
```

- Si può passare al costruttore il percorso completo del file da scrivere, utilizzando le dimensioni del buffer e la codifica predefinite:

```
Dim ScriviStream As New StreamWriter("C:\Nuovofile.txt")
```

- Si può passare al costruttore il percorso completo del file da scrivere ed un parametro, *append*, che

determina se i dati devono essere aggiunti al file. Se il file esiste ed il parametro *append* è *false*, il file viene sovrascritto. Se il file esiste ed il parametro *append* è *true*, i dati vengono aggiunti. Infine, nel caso in cui il file non esiste, il costruttore crea un nuovo file:

```
Dim ScriviStream As New StreamWriter(
                                "C:\Nuovofile.txt", True)
```

La classe *StreamWriter* espone due metodi che permettono di scrivere in un file di testo:

- Il metodo **Write** permette di scrivere la rappresentazione testuale di tutti i tipi primitivi di VB.Net *(Boolean, Integer, Double, ..)*.
- Il metodo **WriteLine**, rispetto al metodo *Write*, aggiunge automaticamente un carattere di nuova riga. Il terminatore di riga predefinito è un ritorno a capo seguito da un avanzamento riga *("\r\n")*.

Possiamo, ora, scrivere il codice che ci permetterà di leggere un file e di crearne uno nuovo, con lo stesso contenuto. Ovviamente avremo bisogno di un oggetto *StreamReader* per leggere il file e di un oggetto *StreamWriter* per scrivere il nuovo file. Il secondo passo sarà quello di ciclare su tutte le righe del file di origine e di scrivere la riga, appena letta, nel file di destinazione. Infine chiuderemo i due oggetti *StreamReader* e *StreamWriter*.

```
Dim LeggiStream As New StreamReader("C:\MioFile.txt")
Dim ScriviStream As New StreamWriter("C:\NuovoFile.txt")
While LeggiStream.Peek <> -1
   ScriviStream.WriteLine(LeggiStream.ReadLine)
End While
LeggiStream.Close()
ScriviStream.Close()
```

LEGGERE E SCRIVERE FILE BINARI

Le classi *BinaryReader* e *BinaryWriter* permettono di leggere e scrivere stringhe e dati di tipo elementare in un file. A differenza di *StreamReader* e *StreamWriter*, le informazioni vengono lette e scritte come dati binari e non come testo, inoltre non si possono creare degli oggetti *BinaryReader* o *BinaryWriter* passando direttamente al costruttore il nome di un file. Per creare, quindi, un oggetto *BinaryReader* o *BinaryWriter*, si deve generare esplicitamente un oggetto *Stream* e passarlo, come argomento, ai metodi costruttori della classe, per questo possiamo scrivere ad esempio:

```
Dim OggettoStream As Stream = File.Open(
         "c:\miofile.txt", FileMode.Open, FileAccess.Read)
Dim LeggiBinario As New BinaryReader(OggettoStream)
```

La classe *BinaryReader* espone numerosi metodi *Read (ReadBoolean, ReadChar, ReadDecimal, ReadDouble….)*, uno per ogni possibile tipo di dato nativo, ed un metodo *PeekChar* che restituisce *-1* nel caso in cui non esistono più caratteri nello stream. Se vogliamo riscrivere la nostra applicazione di esempio utilizzando *BinaryReader*, il codice sarà molto simile a quello scritto in precedenza:

```
'Apertura del file e creazione dell'oggetto BinaryReader
Dim OggettoStream As Stream = File.Open(
    TextBoxPathFile.Text, FileMode.Open, FileAccess.Read)
Dim LeggiBinario As New BinaryReader(OggettoStream)
Dim Contenuto As String
'Itera  fino a che vi   sono dati  a disposizione
While LeggiBinario.PeekChar <> -1
   'Legge l'elemento successivo e concatena il risultato
   Contenuto = Contenuto + LeggiBinario.ReadChar
End While
'chiudiamo gli oggetti aperti
LeggiBinario.Close()
OggettoStream.Close()
'visualizza il risultato nel TextBox
TextBoxRisultato.Text = Contenuto
```

L'utilizzo dell'oggetto *BinaryWriter*, è molto affine all'oggetto *StreamWriter*, poiché il relativo metodo *Write* subisce l'overloading in modo da accettare tutti i tipi primitivi di VB .NET. Se vogliamo riscrivere il codice di esempio che ci permette di leggere un file e di crearne uno nuovo (con lo stesso contenuto), utilizzando *BinaryWriter*, si avrà semplicemente:

```
Dim StreamInLettura As Stream = File.Open(
         "c:\Miofile.txt", FileMode.Open, FileAccess.Read)
Dim StreamInScrittura As Stream = File.Open(
                       "c:\Nuovofile.txt", FileMode.Create,
                    FileAccess.ReadWrite, FileShare.None)
Dim LeggiBinario As New BinaryReader(StreamInLettura)
Dim ScriviBinario As New BinaryWriter(StreamInScrittura)
While LeggiBinario.PeekChar <> -1
   ScriviBinario.Write(LeggiBinario.ReadChar)
End While
LeggiBinario.Close()
ScriviBinario.Close()
```

L'uso di *BinaryWriter* e *BinaryReader* per scrivere e leggere stringhe presenta però alcune problematiche. Quando si passa una stringa al metodo *Write* viene creata una stringa di lunghezza prefissata. Se si vogliono scrivere soltanto i caratteri effettivi, è necessario passare al metodo *Write* una matrice di *Chars*. Per leggere stringhe si possono utilizzare due metodi: il metodo *ReadString*, nel caso di stringhe di lunghezza prefissata oppure il metodo *ReadChar* nel caso di stringhe a lunghezza fissa.

Luigi Buono

FileMode **permette di indicare le modalità di apertura di un file.**

FileAccess **permette di indicare le modalità di accesso in lettura e scrittura di un file.**

FileShare **permette di indicare il tipo di accesso al file di cui dispongono altri thread aperti.**

La classe *ASCIIEncoding* codifica i caratteri Unicode come caratteri ASCII singoli a 7 bit.

La classe *UnicodeEncoding* codifica ciascun carattere Unicode come due byte consecutivi.

La classe *UTF7Encoding* codifica i caratteri Unicode utilizzando la codifica *UTF-7 (UTF-7 indica UCS Transformation Format,* form a 7 bit).

La classe *UTF8Encoding* codifica i caratteri Unicode utilizzando la codifica *UTF-8 (UTF-8 indica UCS Transformation Format,* form a 8 bit).

Attributi e uso dei puntatori nei blocchi di codice unsafe

Attributi e codice unsafe

ultima parte

Giungiamo alla trentesima ed ultima lezione di questo corso dedicato a C#. Oggetto di studio dell'appuntamento finale saranno gli attributi, il codice unsafe ed i puntatori

Questa che state leggendo è l'ultima puntata del corso di C# che ioProgrammo sta pubblicando da oltre due anni. Siamo dunque arrivati agli ultimi argomenti, che rappresentano alcune fra le caratteristiche più peculiari del linguaggio.

ATTRIBUTI

C# prevede la possibilità di aggiungere ulteriori informazioni al codice attraverso degli elementi sintattici chiamati *attributi*. Grazie agli attributi è possibile associare delle informazioni ad una classe, ad un metodo, ad una struttura e via discorrendo. A seconda dello scopo e della funzionalità dell'attributo utilizzato, la struttura sintattica cui ci si riferisce avrà un comportamento diverso in fase di compilazione o di esecuzione. C# dispone di un set di attributi predefiniti, due dei quali saranno esaminati in questa lezione. Il programmatore può poi arricchire il set fornito realizzando degli attributi personalizzati. Tutti gli attributi precedono la struttura sintattica cui si riferiscono, e la loro forma è la seguente:

```
[NomeAttributo(parametri)]
```

L'attributo Conditional

L'attributo *Conditional*, in parole semplici, fornisce una scorciatoia per la compilazione condizionale esaminata nel corso della lezione precedente (direttive *#if, #elif, #else* e *#endif*). Per poter far ricorso all'attributo *Conditional* è necessario importare il namespace *System.Diagnostics* mediante una clausola *using*. L'utilizzo dell'attributo *Conditional* viene dimostrato attraverso il seguente codice esemplificativo:

```
#define PROVA1
#define PROVA2

using System;
using System.Diagnostics;
class Test {
  [Conditional("PROVA1")]
  public void prova1() {
    Console.WriteLine("Sono il metodo prova1");
  }
  [Conditional("PROVA2")]
  public void prova2() {
    Console.WriteLine("Sono il metodo prova2");
  }
  public static void Main() {
    Test t = new Test();
    t.prova1();  // Richiamato solo se PROVA1 è definito
    t.prova2();  // Richiamato solo se PROVA2 è definito
  }
}
```

Il codice definisce due simboli: *PROVA1* e *PROVA2*. Il metodo *prova1()* sarà eseguito soltanto se il simbolo *PROVA1* risulterà definito; alla stessa maniera *prova2()* sarà eseguito solo nel caso in cui sarà stata riscontrata la definizione del simbolo *PROVA2*. Provate pertanto a variare le due definizioni iniziali per verificare il funzionamento dell'attributo *Conditional*.

L'attributo Obsolete

L'attributo *Obsolete* è utile per marcare come obsoleta una struttura del codice. Che utilità può avere questo genere di operazione? Supponiamo di aver realizzato una libreria di

classi utili per compiere una certa serie di operazioni. Queste classi, naturalmente, dispongono di metodi che sono invocati dai programmatori che fanno uso della nostra libreria. A distanza di qualche tempo decidiamo di revisionare la nostra libreria di classi al fine di pubblicarne una nuova versione, con prestazioni migliorate e con nuove caratteristiche aggiunte.
In questo processo di revisione stabiliamo che alcuni metodi presenti nella prima release non dovrebbero più essere utilizzati, magari perché abbiamo cambiato l'approccio di alcune parti della libreria. Rimuovere totalmente i metodi obsoleti è una soluzione errata: la nuova versione della nostra libreria non sarebbe così retrocompatibile con la precedente, e per questo non potrebbe essere usata in corrispondenza di codice scritto prima della sua pubblicazione. Allo stesso tempo, però, si desidera far osservare ai programmatori che fanno uso della libreria di non utilizzare più, d'ora in poi, alcuni dei metodi in essa compresi. A questo scopo è possibile servirsi dell'attributo *Obsolete*.
Il modello cui far riferimento è il seguente:

```
[Obsolete("messaggio")]
```

Ecco un esempio dimostrativo:

```
using System;

class Test {

  [Obsolete("Non usare questo metodo! Usa dividi2")]
  public int dividi(int a, int b) {
    return a / b;
  }

  public int dividi2(int a, int b) {
    if (b == 0) return 0;
    else return a / b;
  }

  public static void Main() {
    Test t = new Test();
    // Alla compilazione mostra un avvertimento.
    Console.WriteLine("10 / 2 = " + t.dividi(10, 2));
    // Nessun avvertimento.
    Console.WriteLine("10 / 2 = " + t.dividi2(10, 2));
  }

}
```

Il metodo *dividi()* è marcato come obsoleto, rimpiazzato da *dividi2()*. Nonostante questo nel *Main()* si fa ricorso a *dividi()*. La compilazione riuscirà tranquillamente, ma il compilatore avrà cura di mostrare in output un messaggio di avvertimento come il seguente:

```
Esempio02.cs(18,37): warning CS0618:
    "Test.dividi(int, int)" è obsoleto: "Non usare questo
                                        metodo! Usa dividi2"
```

CODICE UNSAFE

C# permette l'uso di codice *unsafe*, cioè insicuro. Con la dicitura codice insicuro non si intende del codice mal scritto: la bontà del codice dipende esclusivamente dal programmatore! I compilatori accettano qualunque cosa sia sintatticamente corretta, e purtroppo ancora deve essere inventato un compilatore capace di distinguere un codice ben scritto in ogni particolare da uno insicuro a causa dell'inesperienza del programmatore.
In principio di questo corso si è spiegato come i programmi per .NET siano eseguiti all'interno di uno speciale ambiente chiamato CLR (Common Language Runtime). Il CLR gestisce la memoria, aumentando la robustezza del software realizzato e sollevando il programmatore da alcune gravose responsabilità. Tuttavia, osservando l'altra faccia della medaglia, si capisce che queste buone caratteristiche comportano anche lacune prestazionali e diminuiscono il grado di libertà del programmatore. Benché sia un evento raro, può capitare che si abbia il desiderio o la necessità di escludere parte dei controlli del CLR, soprattutto per tornare a gestire direttamente la memoria come si fa con linguaggi quali C e C++.
Insomma, per farla breve, si può aver bisogno di tornare ai vecchi puntatori, croce e delizia della programmazione di più basso livello. C# permette di farlo, attraverso dei blocchi etichettati come insicuri, proprio perché eseguiti senza la supervisione del CLR. La responsabilità del buon uso della memoria, nei blocchi unsafe, cade di nuovo nelle mani del programmatore.

UTILIZZO DEI PUNTATORI

Questo paragrafo non ha la presunzione di voler spiegare in poche righe come si faccia uso dei puntatori a chi non ne ha mai sentito parlare; piuttosto si presenta come un prontuario per chi già ha programmato con altri linguaggi che ne fanno uso abbondante.

- **GUIDA A C#**
  Herbert Schildt
  (McGraw-Hill)
  ISBN 88-386-4264-8
  2002

- **INTRODUZIONE A C#**
  Eric Gunnerson
  (Mondadori Informatica)
  ISBN 88-8331-185-X
  2001

- **C# GUIDA PER LO SVILUPPATORE**
  ***Simon Robinson* e altri**
  (Hoepli)
  ISBN 88-203-2962-X
  2001

Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile. Se la variabile a contiene l'indirizzo della variabile b si dice che "a punta a b". I puntatori si dichiarano al seguente modo:

```
tipo* nome
```

Ad esempio:

```
int* a
```

In un caso come quello appena mostrato il puntatore a potrà memorizzare l'indirizzo di una qualsiasi variabile di tipo *int*. È molto importante osservare che solo i tipi non-classe possono essere puntati in questa maniera.
L'operatore unario **&** può ricavare l'indirizzo di una variabile esistente, in modo che lo stesso possa poi essere utilizzato, ad esempio proprio per memorizzarlo all'interno di un puntatore. Per questo motivo l'operatore **&** viene letto come "indirizzo di".
Ecco un esempio:

```
int a = 5;
int* p = &a;
```

In questo modo si otterrà un puntatore *p* che punterà alla variabile *a*. L'operatore unario ***** è complementare rispetto a **&**. Permette la lettura del valore memorizzato nella variabile puntata da un indirizzo di memoria. Per questo motivo l'operatore viene letto come "valore di".
Proseguendo l'esempio precedente:

```
int a = 5;
int* p = &a;
int b = *p;
```

In questa maniera la variabile b riceve il valore della variabile puntata da *p*, cioè *5*.
I puntatori possono essere utilizzati anche in corrispondenza delle strutture:

```
struct Rettangolo {
  public int larghezza;
  public int altezza;
  public int area() { return larghezza * altezza; }
}
Rettangolo r = new Rettangolo();
Rettangolo* p = &r;
```

I singoli campi di una struttura raggiunta mediante puntatore possono essere selezionati servendosi dell'operatore **->**, al posto dell'operatore **.** (punto) utilizzato con le normali variabili:

```
p->larghezza = 5;
p->altezza = 3;
int area = p->area();
```

LA CLAUSOLA UNSAFE

Per utilizzare i puntatori di C# è necessario specificare esplicitamente che si sta scrivendo del codice potenzialmente insicuro.
La parola chiave necessaria è *unsafe*, che può essere utilizzata per etichettare una singola istruzione, un blocco oppure un intero metodo:

```
unsafe istruzione;

unsafe {
  // blocco
}
unsafe public void metodo() {
  // metodo
}
```

Ecco un esempio:

```
using System;

class Test {
  unsafe public static void Main() {
    int a = 5;
    int* p = &a;
    *p += 2;
    Console.WriteLine(a);
  }
}
```

I programmi che contengono delle porzioni unsafe vanno compilati al seguente modo:

```
csc /unsafe NomeFile.cs
```

Tutte queste accortezze servono per verificare che il programmatore sia realmente cosciente di quel che sta facendo.

LA CLAUSOLA FIXED

C# dispone dunque di due differenti tecniche per la gestione della memoria: quella automatica, attuata dal CLR, e quella "manuale", svolta dal programmatore attraverso i puntatori ed i blocchi di codice unsafe. Può capita-

re che le due tecniche vadano in conflitto, per via della garbage collection svolta dal CLR. Si ricorda che la garbage collection è il procedimento messo in atto dal CLR per ripulire la memoria dagli oggetti non più utilizzati e per ricompattare quelli rimanenti. Non è possibile sapere quando il CLR avvierà la garbage collection. Per questo motivo può capitare che un puntatore rivolto al campo di un oggetto perda la propria consistenza.
Supponiamo di avere la seguente classe:

```
class Rettangolo {
  public int larghezza;
  public int altezza;
  public int area() { return larghezza * altezza; }
}
```

Inoltre, supponiamo poi di avere da qualche parte un codice come il seguente, che fa uso di puntatori:

```
Rettangolo r = new Rettangolo();
int* p = &r.larghezza;
// ...
*p += 6;
```

Può accadere che la garbage collection venga eseguita dopo la seconda istruzione, all'altezza del commento che vuole rappresentare l'eventuale presenza di altro codice non relazionato agli elementi mostrati. A seguito della garbage collection e della ricompattazione della memoria può accadere che l'oggetto *r* non si trovi più nella stessa area di memoria in cui era inizialmente. I puntatori, in un caso come questo, non vengono aggiornati automaticamente. Allora *p* si ritroverà a puntare un'area della memoria che non corrisponde più al campo larghezza dell'oggetto *r*. Naturalmente questo è un grave problema, sia perché il valore di *p* non può più essere letto con certezza di corrispondeza sia perché aggiornando lo stesso si corre il rischio di compromettere altri dati che dopo la garbage collection sono andati ad occupare la porzione di memoria che prima era dell'oggetto *r*. C# pone rimedio a questo problema con la clausola *fixed*, che permette di fissare un oggetto in memoria. Quando un oggetto è contrassegnato come fisso la garbage collection non lo sposterà per nessuna ragione dalla sua posizione in memoria. In questo modo i puntatori manterranno sempre la corretta corrispondenza. L'uso della clausola avviene alla seguente maniera:

```
fixed (tipo* p = &oggetto.campo) {
  // codice che fa uso di p
}
```

Se la garbage collection dovesse essere lanciata durante l'esecuzione delle istruzioni presenti nel blocco, l'oggetto associato al puntatore non sarà spostato. Ecco un esempio pratico:

```
using System;

class Rettangolo {
  public int larghezza;
  public int altezza;
  public int area() { return larghezza * altezza; }
}

class Test {
  unsafe public static void Main() {
    Rettangolo r = new Rettangolo();
    r.larghezza = 7;
    r.altezza = 3;
    fixed (int* p = &r.larghezza) {
      *p -= 2;
    }
    Console.WriteLine(r.area());
  }

}
```

Ricordatevi di utilizzare sempre la clausola *fixed* ogni volta che vi troverete a gestire insieme puntatori ed oggetti. Gli errori derivati dal mancato utilizzo di *fixed* sono infatti difficili da scovare, proprio perché capitano in situazioni particolari e rare. Il fatto che l'occorrere di un errore sia raro, ad ogni modo, non rende l'errore stesso meno grave.
E poi, si sa: se qualcosa può andare storto lo farà sempre al momento meno opportuno.

CONCLUSIONI

Siamo così giunti al termine dell'ultima lezione di questo corso. Mi auguro che lo abbiate seguito con piacere ed interesse, e vi auguro di diventare degli ottimi programmatori C#. Di strada da fare ce n'è ancora molta, ma con solide nozioni di base apprendere le tecniche più avanzate è semplice e divertente. Dunque buon divertimento e alla prossima, ricordatevi di non perdete i prossimi numeri di ioProgrammo, dove gli aspetti più pratici di C# troveranno lo spazio sufficiente per consentirvi il proseguimento dello studio intrapreso.

Carlo Pelliccia

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

Lanciare e gestire le eccezioni in Java

Applicazioni più robuste

Uno dei punti di forza di Java è un sistema robusto e affidabile per gestire gli errori. Questo mese imparerai cosa sono le eccezioni e come usarle: una tappa fondamentale verso applicazioni "robuste"

❑ CD ☑ WEB
CorsoJava_85.zip
cdrom.ioprogrammo.it

La gestione degli errori è una parte importante di Java, e non puoi usare il linguaggio se non conosci questi meccanismi. Questo articolo introdurrà l'argomento, e quello del mese venturo lo approfondirà. Ecco cosa ti aspetta:

- Imparerai ad usare la parola chiave *super*.
- Scoprirai cos'è un'eccezione.
- Imparerai a lanciare e gestire le eccezioni.

Prima di entrare nel vivo di questo articolo, però, voglio spiegarti qualcosa in più su un argomento di qualche mese fa: i costruttori.

RITORNO AI COSTRUTTORI

Guarda questa coppia di classi:

```
public class ClasseBase {
    protected int x;
    public ClasseBase() { x = 10;
        System.out.println("ClasseBase()");}}
public class ClasseDerivata extends ClasseBase {
    public ClasseDerivata() {
        System.out.println("ClasseDerivata()");
        System.out.println("x vale " + x);}
    public static void main(String[] args) {
        new ClasseDerivata();} }
```

Cosa succede se lanci *ClasseDerivata.main()*?

```
ClasseBase()
ClasseDerivata()
x vale 10
```

Quando costruisci un oggetto, Java cerca prima di chiamare il costruttore della sua superclasse. Se non lo facesse, non avresti la garanzia che l'oggetto sia costruito correttamente. Nel nostro esempio, *ClasseDerivata* eredita il campo *x* di *ClasseBase*. Se il costruttore di *ClasseBase* non venisse chiamato, questo campo non verrebbe inizializzato. Dato che *ClasseDerivata* potrebbe voler accedere a *x* (e in effetti lo fa), il costruttore di *ClasseBase* deve essere chiamato il più presto possibile, prima di qualsiasi operazione sulla classe derivata. Java ottiene questo risultato aggiungendo all'inizio del costruttore di *ClasseDerivata* una chiamata *"invisibile"* al costruttore di *ClasseBase*. Se vuoi, puoi fare la stessa cosa in modo esplicito usando la parola chiave *super():*

```
public class ClasseDerivata...
    public ClasseDerivata() {
        super();
        System.out.println("ClasseDerivata()");
        System.out.println("x vale " + x);}
```

La chiamata a *super* deve sempre essere la prima operazione di un costruttore, pena un errore di compilazione. Se non scrivi esplicitamente *super(),* Java aggiunge questa chiamata per conto proprio. Tutto questo avviene sempre, a patto che che la superclasse abbia un costruttore senza argomenti (ti ricordo che se non scrivi il costruttore, Java ne genera silenziosamente uno senza argomenti e vuoto). Ora proviamo a sostituire il costruttore senza argomenti di *ClasseBase* con uno che prende un argomento intero:

```
public class ClasseBase {
    private int x;
    public ClasseBase(int parametro) {
        x = parametro;
        System.out.println("ClasseBase(" + parametro + ")");
    } }
```

A questo punto, *ClasseDerivata* non compila più. Il compilatore genera un errore simile a questo:

REQUISITI

Conoscenze richieste
Concetti di ereditarietà e polimorfismo

Software
Java 2 Standard Edition SDK 1.4 o superiore.

Impegno

Tempo di realizzazione

```
ClasseBase(int) in ClasseBase cannot be applied to ()
```

Proprio come prima, Java esige che il costruttore della *ClasseBase* venga chiamato all'inizio del costruttore della *ClasseDerivata.* Ma questa volta il costruttore della *ClasseBase* richiede un argomento intero. Quindi Java non può invocare automaticamente il *super(),* perché non saprebbe quale valore dare a questo argomento. In casi come questo ci tocca chiamare esplicitamente il *super()* passandogli un valore intero:

```
public class ClasseDerivata...
   public ClasseDerivata() {
      super(10);
      System.out.println("ClasseDerivata(
                              " + parametro + ")");}
```

Qui ho passato un valore qualsiasi, ma di solito è meglio lasciare che sia il client a decidere il valore dei parametri:

```
public class ClasseDerivata...
   public ClasseDerivata(int parametro) {
      super(parametro);
      System.out.println("ClasseDerivata(
                              " + parametro + ")");}
   public static void main(String[] args) {
      new ClasseDerivata(10);}
```

Il risultato è:

ClasseBase(10)
ClasseDerivata(10)
x vale 10

Riassumiamo:

- Java vuole che il costruttore della classe base sia sempre chiamato per primo nel momento in cui si crea un oggetto della classe derivata;
- se la classe base ha un costruttore senza argomenti, Java aggiunge la chiamata automaticamente.
- se la classe base non ha un costruttore senza argomenti, o se la classe base ha più costruttori e vuoi decidere tu quale chiamare, allora devi farlo esplicitamente con la parola chiave super.
- *super()* deve essere sempre la prima istruzione nel costruttore della classe derivata.

Tutto questo tornerà utile tra poco.

BUONI ARGOMENTI

Prima del nostro esempio, ti ricordo che la Java Virtual Machine passa gli argomenti della riga di comando al *main()* sotto forma di un array di stringhe. **Esercizio 1: Scrivi un programma che stampa gli argomenti della sua riga di comando.** Forse hai già risolto questo esercizio qualche mese fa. Ecco la soluzione:

```
public class Argomenti {
   public static void main(String[] args) {
      for(int i = 0; i < args.length; i++)
         System.out.println(args[i]);} }
```

Prova a lanciare questo programma con qualche argomento:

```
java it.ioprogrammo.corsojava.eccezioni.Argomenti a b c def
```

Il risultato è in Fig. 1.

LINGUE SCONOSCIUTE

Ecco finalmente il primo esercizio “corposo” di questo mese. **Esercizio 2: Scrivi un programma che prende una stringa dalla riga di comando, converte tutte le vocali in 'i' e stampa la stringa sullo schermo.** La mia soluzione:

Fig. 1: ***I parametri sono restituiti correttamente***

```
package it.ioprogrammo.corsojava.eccezioni1;
public class Cinesizzatore {
   public static void main(String[] args) {
      String originale = args[0];
      System.out.println(cinesizza(originale));}
   private static String cinesizza(String originale) {
      String cinese = "";
      for(int i = 0; i <= originale.length(); i++)
         cinese += cinesizza(originale.charAt(i));
      return cinese;}
   private static char cinesizza(char c) {
      if(c == 'a' || c == 'e' || c == 'o' || c == 'u')
         return 'i';
      return c;} }
```

Il programma usa il metodo statico *overloaded cinesizza()* per convertire tutte le vocali in *'i'*. Una delle due varianti del metodo converte un singolo carattere, e l'altra usa la prima variante per convertire tutti i caratteri di una stringa. **Esercizio 3: La mia soluzione all'esercizio 1 ha diversi problemi. Trovane due.** Ho usato due metodi della classe *String:* il metodo *String.length()* restituisce la lunghezza della stringa; il metodo *String. charAt(i)* restituisce il carattere di indice i delle stringa. Il primo problema salta fuori appena lanci il programma:

```
java it.ioprogrammo.corsojava.eccezioni.Cinesizzatore
                              "Non è bello ciò che è bello"
```

Ecco lo sconfortante risultato:

```
java.lang.StringIndexOutOfBoundsException: String index
   out of range: 27 at java.lang.String.charAt(Unknown
         Source) at it.ioprogrammo.corsojava.eccezioni.
           Cinesizzatore.cinesizza(Cinesizzatore.java:13)
   at it.ioprogrammo.corsojava.eccezioni.
                  Cinesizzatore.main(Cinesizzatore.java:7)
Exception in thread "main"
```

Purtroppo il mio codice contiene un bug, un errore di programmazione. Il metodo *String.charAt()* usa le stesse convenzioni di un array: l'indice va da 0 alla lunghezza della stringa meno 1. La stringa dell'esempio è lunga 27 caratteri, quindi gli indici vanno da 0 a 26. Ma nel ciclo ho usato un "maggiore e uguale" al posto di un "maggiore", quindi nell'ultima iterazione il programma cerca di referenziare il carattere di indice 27. Se fossi stato un po' più attento quando scrivevo il codice, non sarei finito in questa situazione imbarazzante. Capita. Se avessi scritto il programma in un altro linguaggio, forse avrei rischiato di leggere un valore a casaccio dalla memoria. Per fortuna Java controlla tutti gli accessi indicizzati alle stringhe, quindi si accorge dell'errore e gestisce la cosa in modo elegante: lancia un'eccezione. Un'eccezione è un oggetto che segnala che qualcosa è andato storto durante l'esecuzione del programma. Le eccezioni sono istanze della classe *Exception,* o di qualche sua sottoclasse. In particolare questo mese parleremo delle eccezioni *unchecked*, che sono istanze di *RuntimeException,* che a sua volta è una sottoclasse di *Exception.* Nel nostro caso, una *StringIndexOutOfBoundsException* (una sottoclasse di *RuntimeException*) indica che il programma cerca di accedere fuori dai limiti della stringa. L'eccezione non si limita a segnalare il tipo di errore: dice anche dove si è verificato (alla riga 13 del *Cinesizzatore*), e quale è stata la sequenza di chiamate che lo ha generato (è successo nel metodo *String.charAt()*, chiamato a sua volta dalla riga 13 del metodo *Cinesizzatore.cinesizza(),* che era stato chiamato dalla riga 7 del metodo *Cinesizzatore .main()*). In più l'eccezione indica che il programma ha cercato di accedere al carattere di indice 27. Grazie a tutte queste informazioni posso capire facilmente dove ho sbagliato, e correggere il mio errore:

```
public class Cinesizzatore...
  private static String cinesizza(String originale)
  {   String cinese = "";
    for(int i = 0; i < originale.length(); i++)
      cinese += cinesizza(originale.charAt(i));
    return cinese; }
```

Modifica il Cinesizzatore in modo che converta correttamente le vocali maiuscole (compresa la 'e' maiuscola accentata, che è un carattere a sè stante).

IL GIOCO DELLA STAFFETTA

Un'eccezione è come una palla, che una riga di programma (o la Virtual Machine in persona) lancia per segnalare un problema. Quando un metodo incontra un'eccezione, il normale flusso del programma si interrompe, e il controllo torna al client che ha chiamato il metodo. Il chiamante potrebbe gestire l'eccezione (presto vedrai come). In caso contrario, anche il chiamante si interrompe e a sua volta passa la palla al proprio chiamante, e così via. Come una patata bollente, l'eccezione passa di mano in mano fin quando qualcuno non la gestisce, o fin quando il controllo non passa al primo di tutti i client: la Virtual Machine che ha chiamato il *main().* Proviamo a inventare un tipo di eccezione tutto nostro e a lanciarla. Come ti ho detto, tutte le eccezioni derivano da *Exception,* ma questo mese ci occuperemo in particolare di quelle che derivano da *RuntimeException,* a sua volta una sottoclasse di *Exception* (il mese venturo parleremo più in dettaglio della gerarchia delle eccezioni):

```
public class SimpaticaEccezione extends RuntimeException {
  public SimpaticaEccezione(String msg) {
    super(msg); } }
```

RuntimeException ha più costruttori, incluso uno senza argomenti. Nel nostro caso ho invece chiamato, con *super(),* il costruttore che prende una String. Questa stringa è un messaggio associato all'eccezione. In questo modo l'eccezione può contenere, oltre alla descrizione generica del problema (la sua classe) anche eventuali dettagli specifici (il messaggio). Puoi lanciare un'eccezione con la parola chiave *throw.* Di solito si crea l'eccezione nel punto in cui la si lancia:

```
throw new RuntimeException("messaggio");
```

Ecco un esempio:

```
package it.ioprogrammo.corsojava.eccezioni1;
public class Staffetta {
  public static void main(String[] args) {
    a();
    System.out.println("questa istruzione non viene
                                  mai eseguita"); }
  private static void a() {
    b();
    System.out.println("questa istruzione non viene
                                   mai eseguita");}
  private static void b() {
    lanciaEccezione();
    System.out.println("questa istruzione non viene
                                   mai eseguita");}
  private static void lanciaEccezione() {
    throw new SimpaticaEccezione("eccomi qua!");
    //System.out.println("questa istruzione non viene
                    mai eseguita"); // non compila! }
}
```

Se fai girare questo programma, otterrai:

```
it.ioprogrammo.corsojava.eccezioni1.SimpaticaEccezione:
                                            eccomi qua!
   at it.ioprogrammo.corsojava.eccezioni1.Staffetta.
                         lanciaEccezione(Staffetta.java:18)
   at it.ioprogrammo.corsojava.eccezioni1.Staffetta.b(
                                        Staffetta.java:14)
   at it.ioprogrammo.corsojava.eccezioni1.Staffetta.a(
                                        Staffetta.java:10)
   at it.ioprogrammo.corsojava.eccezioni1.Staffetta.main(
                                         Staffetta.java:6)
Exception in thread "main"
```

Il *main()* ha chiamato il metodo *a()*. Quando il sistema entra in *a()*, da qualche parte deve esserci scritto chi lo ha chiamato - altrimenti, all'uscita da *a()*, il codice non saprebbe dove tornare. Lo stesso avviene quando *a()* chiama *b()*: il sistema deve ricordare che *b()* è stato chiamato da *a()*, e che *a()* è stato chiamato da *main()*. Quindi, in ogni momento, mentre il programma gira, esiste una struttura che "ricorda" l'ordine di tutte le chiamate, dalla JVM che chiama il *main()* fino al metodo attualmente in esecuzione. Questa struttura si chiama *stack* delle chiamate a metodo. Il metodo *b()* chiama *lanciaEccezione()*, che crea e lancia una *SimpaticaEccezione*. A questo punto è come se il sistema incontrasse una forma anomala di *return*: il metodo termina immediatamente, e il controllo torna al metodo chiamante *b()*. Ma anche *b()*, non sapendo cosa fare dell'eccezione, termina immediatamente e passa l'eccezione al proprio chiamante *a()*. Come una staffetta, l'eccezione passa di mano in mano, facendo il percorso inverso delle chiamate a metodo, fin quando non torna al main() e infine alla JVM. Questo segna la fine del programma. Nessuna delle *System.out.println()* viene eseguita, perché l'eccezione interrompe il flusso del programma e costringe Java a "riavvolgere il nastro". Ho dovuto persino commentare la stampa nel metodo *lanciaEccezione()* perché il compilatore, accorgendosi che quel codice è in ogni caso irraggiungibile, avrebbe altrimenti rifiutato il programma.

AFFERRALE AL VOLO

Abbiamo detto che le eccezioni si propagano all'indietro attraverso lo stack delle chiamate ai metodi, fin quando non raggiungono la radice del programma. Ma perché questo lungo viaggio? Non sarebbe meglio se la Virtual Machine fermasse tutto e stampasse l'eccezione sullo schermo? Il motivo è che uno dei metodi sulla strada dell'eccezione potrebbe sapere come rimediare al problema, o per lo meno potrebbe mettere in atto qualche misura di emergenza – forse segnalare il problema all'utente, o scriverlo su un file di log, o riprovare l'operazione con parametri diversi. Tutte queste sono strategie per gestire l'eccezione. L'idea è che non sempre il codice che incontra un'eccezione sa cosa farne; ma forse qualche altro metodo, nella catena dei chiamanti, lo sa. Per gestire un'eccezione la devi "afferrare", durante il suo viaggio all'indietro verso il *main()*, con le parole chiave *try* e *catch* (che letteralmente significano prova e afferra). Queste parole si usano sempre insieme in un doppio blocco di istruzioni chiamato blocco *try-catch:*

```
try {
  codiceChePotrebbeGenerareEccezioni();
} catch (ClasseDiEccezioni nomeEccezione) {
  codiceCheGestisceLEccezione();}
```

Il blocco *try* prova ad eseguire il codice che potrebbe lanciare un'eccezione. Se si verifica un'eccezione del tipo specificato, allora il controllo passa al blocco catch, che prenderà gli opportuni provvedimenti. Ad esempio:

```
public class TryCatch {
    public static void main(String[] args) { a();}
    private static void a() {
        try { b(); }
        catch (SimpaticaEccezione e) {
            System.out.println("SimpaticaEccezione in
                                              TryCatch.a()");
            System.out.println("Messaggio:" + e.getMessage()); }
        System.out.println("Questa riga viene eseguita");}
    private static void b() { lanciaEccezione(); }
    private static void lanciaEccezione() {
        throw new SimpaticaEccezione("eccomi qua!");
    }
}
```

Il *main()* chiama *a()*, che chiama *b()*, che chiama *lanciaEccezione()*, che lancia una *SimpaticaEccezione*. Ma nel suo viaggio all'indietro verso il *main()*, l'eccezione si ritrova all'interno di un blocco *try-catch*. Il controllo passa subito al blocco *catch*, che dichiara di voler afferrare tutte le eccezioni di classe *SimpaticaEccezione*. Qui l'eccezione riceve un nome (come se fosse un parametro) e viene usata per le eventuali contromisure. L'esempio si limita a stampare qualche messaggio sullo schermo (il metodo *Exception.getMessage()* restituisce il messaggio associato all'eccezione):

SimpaticaEccezione in TryCatch.a()
Messaggio: eccomi qua!
Questa riga viene eseguita

Per questo mese ci fermiamo qui. Il mese venturo faremo altri esempi, e parleremo delle importanti eccezioni *checked*. A presto!

Paolo Perrotta

RIVOLGERSI ALLA FAMIGLIA

Puoi usare super, seguita dall'invocazione di un metodo, anche per chiamare un metodo della superclasse che non è un costruttore:

```
super.metodo()
```

Questa forma è utile quando devi chiamare la versione originale di un metodo dalla sua versione "overridden":

```
class Superclasse {
 public void f() {
  System.out.println(
        "Superclasse.f()");
  faiQualcosaDiImportante();
 }
}
class Sottoclasse extends
                Superclasse {
 public void f() {
  System.out.println(
        "Sottoclasse.f()");
  super.f(); // chiama
              Superclasse.f()
   faiAltreCose(); }
}
```

***Sottoclasse.f()* aggiunge funzionalità a *Superclasse.f()* senza eliminare le funzionalità originali e senza duplicare il codice. Quando usi super in questo modo, non sei nemmeno obbligato ad usarlo come prima istruzione del metodo.**

Applicare lo stile di Windows XP all'interfaccia di un'applicazione

Interfacce in stile XP

Dopo una breve panoramica su Windows XP Visual Style implementeremo un'applicazione che interagisce con un database Access. Un look nuovo per i nostri programmi

Le Visual Styles sono le specifiche relative all'aspetto degli elementi dell'interfaccia utente. Esse definiscono i colori, i font, le dimensioni dei controlli ecc. e forniscono i meccanismi per impostare le interfacce delle applicazioni Windows-based. Faremo una panoramica sullo stile Windows XP e sulle problematiche connesse alla sua applicazione nei progetti Visual Basic e descriveremo un'applicazione (naturalmente in stile Windows XP) che amministra i dati dei clienti e dei relativi ordini. L'applicazione interagisce con il database *ordini.mdb* che ha due tabelle: *Clienti* e *Ordini*. Con Windows XP sono fornite la versione 5 e 6 della *ComCtl32,* alla base del nuovo look e feel, la versione 6, rispetto alle versioni precedenti, non può essere distribuita insieme al progetto Visual Basic, quindi un'applicazione potrà usarla solo se è installata nel sistema operativo. Un'applicazione Visual Basic, di default, usa gli *user control* definiti in *User32.dll* e i common control definiti in *ComCtl32.dll* versione 5; in questo appuntamento descriveremo come, predisponendo un file di configurazione (un'application Manifest) un'applicazione Visual Basic riesce ad utilizzare gli *user control* ed i *common control* definiti in *ComCtl32.dll* versione 6. Facciamo notare che in realtà, eseguendo un'applicazione Visual Basic su Windows XP (con tema Windows XP), la *ComCtl32* versione 6 già di default applica lo stile XP alla non-client area del form (cioè al frame, alla caption e alla non-client scroll bars). Con la tecnica che ora descriveremo, estenderemo lo stile XP anche ai controlli dell'area client.

REQUISITI

Conoscenze richieste
Elementi di Visual Basic

Software
Piattaforma Windows XP - Visual Basic 6 SP6.

Impegno

Tempo di realizzazione

MANIFEST PER WINDOWS XP

I *Manifest* sono una parte della tecnologia introdotta da Microsoft per risolvere il problema dei conflitti tra le versioni della stessa DLL. Un Manifest può contenere i metadati che nei precedenti sistemi operativi erano inseriti nel registro di sistema, cioè informazioni sulle *COM Class*, sulle interfacce e sulle librerie. Un file *Manifest* può descrivere un *Side-by-Side assembly* (una collezione di risorse per esempio una DLL) o una isolated application. Nel nostro caso il file Manifest, serve per specificare che un'applicazione deve agganciarsi alla versione 6 della *ComCtl32.dll* (se installata). Di seguito riportiamo il file *Manifest* che useremo nei nostri esempi:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:
                    asm.v1" manifestVersion="1.0">
<assemblyIdentity
   version="1.0.0.0"
   processorArchitecture="X86"
   name="nomesocieta.nomeprodotto.nomeapplicazione"
   type="win32" />
<description>descrizione applicazione</description>
<dependency>
  <dependentAssembly>
    <assemblyIdentity
       type="win32"
       name="Microsoft.Windows.Common-Controls"
       version="6.0.0.0"
       processorArchitecture="X86"
       publicKeyToken="6595b64144ccf1df"
       language="*" />
  </dependentAssembly>
</dependency>
</assembly>
```

Il codice precedente va salvato in un file testo, con estensione XML, e posto nella directory che contiene il file *EXE* dell'applicazione che deve essere trasformata in stile XP. Ma, attenzione: il nome del file *Manifest* deve essere nel seguente formato: *nomevostraapplicazione.exe.manifest.* Nelle *Tabelle 1* e *2* sono riassunte le parti principali del file *Manifest.*

IDE IN STILE XP

Quello che abbiamo detto, in merito al file *Manifest,* lo potete testare impostando, in stile XP, parte dell'IDE di Visual Basic. Per fare ciò basta creare il file *vb6.exe.manifest* e salvarlo nella direc-

tory che contiene il file *vb6.exe*. Dopo aver fatto ciò i controlli intrinseci di Visual Basic, trascinati su un form, risulteranno in stile XP. Lo stile XP però, in questo caso, è applicato solo in fase di progettazione, non resta traccia nel compilato.
Vb6.exe.manifest, dunque, è utile solo in fase di progettazione mentre, per applicare lo stile XP a una vostra applicazione, fate così:

1. Creare un file *Manifest* il cui nome rispetti il formato *nomeapplicazione.exe.manifest.*
2. Utilizzare la funzione *InitCommonControls,* come prima istruzione della vostra applicazione (per comunicare al sistema che si vuole utilizzare la libreria *ComCtl32*).

CARICARE LA COMCTL32

Il secondo passo per usare lo stile XP, dunque, è l'inserimento delle seguenti istruzioni nel vostro progetto:

```
Private Declare Function InitCommonControls
                              Lib "Comctl32.dll" () As Long
Private Sub Form_Initialize()
  Call InitCommonControls
End Sub
```

Cioè la dichiarazione delle funzione *InitCommonControls* e la sua invocazione. Facciamo notare che la *InitCommonControls* deve essere invocata nella *Initialize* altrimenti il form non potrà essere caricato! Allora, se "siete fortunati", per usare lo stile XP oltre a prevedere il file *Manifest* sarà necessario aggiungere le istruzioni per caricare la *ComCtl32.dll.* Se invece la fortuna non vi assiste, dovete leggere attentamente il paragrafo successivo!

CONTROLLI INTRINSECI E STILE XP

I controlli intrinseci *OptionButton* e *Frame* creano alcuni problemi, quando sono usati con lo stile XP. Provate ad inserire in un *Frame* un *OptionButton* o un altro *Frame*. In fase di esecuzione vedrete l'*OptionButton* di colore nero e il *Frame* interno con la label diversa da quella impostata. Questi problemi possono essere risolti inserendo i controlli, interni al frame, in un controllo *PictureBox*. In Fig. 1 mostriamo il problema e il modo in cui è stato risolto (notate che le PictureBox sono state impostate senza bordo e dello stesso colore del frame). Per riprodurre il form di Fig. 1, creare un nuovo progetto (per esempio *Progetto1*), predisponete i controlli intrinseci (con e senza *PictureBox*), impostate il file Manifest (*progetto1.exe.manifest*) nella directory del progetto, compilate e lanciate *progetto1.exe*. Ora possiamo introdurre l'applicazione d'esempio.

GESTIONE ORDINE

L'applicazione di esempio (forse con un po' di presunzione) è stata chiamata *Gestione Ordini*. Essa è

Attributo	Descrizione
Version	Versione del Manifest. La versione deve essere nel seguente formato: *n.n.n.n* con *n<=65535*
ProcessorArchitecture	Tipo di processore utilizzato, di solito X86
Name	Dovrebbe includere il nome della compagnia, il nome del prodotto e il nome dell'applicazione (per esempio *Microsoft.Windows.Applicazione*).
Type	Il tipo di applicazione, di solito Win32

Tabella 1: Parti principali che compongono il file Manifest

Attributo	Descrizione
Type	Il tipo di componente da cui dipende l'applicazione, di solito Win32
Name	Il nome del componente
Versione	Versione del componente
processorArchitecture	*X86*
publicKeyToken	*Key* pubblica usata per firmare il componente, 8 byte cioè 16 caratteri esadecimali
Language	Lingua del componente, di solito "*" per non specificare nessun lingua

Tabella 2: Altre componenti del file Manifest

l'interfaccia per un database Access *(ordini .mdb)* che include le tabelle *Clienti* (con i campi *ClienteId, Nomesocietà, Città, CAP, Paese*) e *Ordini* (con i campi *OrdineId, ClienteId, Note, Npezzi, Prezzo*). La chiave della tabella *Clienti* è il campo *ClientId,* mentre della tabella *Ordini* sono i campi *OrdiniId* e *ClientId,* in base a ciò, ad ogni cliente corrispondono più ordini ed un ordine (per semplificare) è composto da una sola riga. Sul database, per semplificare ulteriormente, è necessario impostare una relazione (join) tra le tabelle che "includi tutti i record di *Ordini* e solo i record di *Clienti* in cui i campi collegati sono uguali". Per quanto riguarda il progetto Visual Basic includere le librerie *ADO Data Control 6.0* (dato che utilizzeremo un controllo ADO data), *Windows Common Control 5.0* e *Windows Common Control-2 5.0.* Sul form, invece, inseriamo un *TabStrip* con due schede: *Imposta* e *Ricerca.* La *Imposta* deve contenere gli elementi che permettono di inserire un nuovo *Cliente* oppure avviare la ricerca dei suoi *Ordini;* la scheda *Ricerca,* invece, su un *ListView,* deve mostrare gli ordini trovati nel database (facciamo notare che manca la scheda per gestire gli ordini e che le PictureBox contenitore le abbiamo disposte ma-

Fig. 1: Gli errori nello stile XP con i controlli intrinseci

nualmente sulla TabStrip…). Ricordiamo che le schede del TabStrip non sono oggetti contenitore (come nel caso dell'SSTab) per questo ad ogni scheda bisogna associare una PictureBox che raggruppa i controlli che deve mostrare. Abbiamo previsto un Array di PictureBox, cioè *Picture1(0)* per la scheda *Imposta* e *Picture1(1)* per la scheda *Ricerca* (ecc. se ci fossero altre schede). In dettaglio i controlli da prevedere per la scheda *Imposta* (cioè sulla *Picture1(0)*) sono: un frame con all'interno gli oggetti (Textbox, Label) associati ai campi della tabella *Clienti* e un pulsante (all'interno di una PictureBox!) per avviare la ricerca degli ordini (nominato *trova*); all'esterno del frame bisogna prevedere due pulsanti (*nuovo* e *cancella*) e un controllo *ADO data*. Sulla *Scheda Ricerca*, invece, bisogna predisporre un ListView per presentare i dati degli ordini del cliente (specificato sulla scheda *Imposta*). Inoltre, all'esterno del TabStrip bisogna disporre un controllo *UpDown* per selezionare le schede. In conformità a quello che abbiamo descritto in precedenza e alle librerie che abbiamo incluso s'intuisce che il nostro progetto è, quasi, compatibile con lo stile XP, l'unico problema, infatti, è costituito dall'ADO Data che non cambia style in XP. Sappiamo, però che con qualche pulsante e qualche riga di codice si può nascondere o evitare l'*ADO Data*. Le due schede in stile XP sono presentate nelle Fig. 2 e 3. Ora possiamo passare alla descrizione del codice.

Fig. 2: *Il form in stile XP*

Fig. 3: *La scheda Ricerca, del TabStrip, in stile XP*

TEMI WINDOWS XP

Un tema (visual style) in un sistema operativo Microsoft determina lo sfondo, lo screen saver, i tipi di carattere, i colori ecc. In Windows XP di default sono inclusi due visual style: Windows XP e Windows Classico (cioè lo stile Windows 95). Ricordiamo che i temi si selezionano dal pannello di controllo utilizzando lo strumento schermo e che altri temi sono forniti in Windows XP Plus Pack.

INTERAGIRE CON IL DATABASE

Innanzi tutto vediamo il codice necessario per interagire con il database. Per caricare l'*ADO Data* e i textbox della scheda *Imposta* utilizziamo la seguente:

```
Sub CaricaControlloAdoCliente()
 Adodc1.ConnectionString = " Provider=
                    Microsoft.Jet.OLEDB.4.0;" _
      & "Data Source=" + App.Path + "\ordini.mdb;" _
                  & "Persist Security Info=False"
 Adodc1.RecordSource = "clienti"
 Adodc1.Refresh
 Set Me.txtcodice.DataSource = Adodc1
 Me.txtcodice.DataField = "clienteid"
…
 Me.Txtcitta.DataField = "citta"
End Sub
```

Notate che leghiamo i TextBox, con la proprietà *DataSource*, all'*ADO Data* così le azioni fatte sui TextBox si riflettono sul database. Per quanto riguarda gli *Ordini*, relativi ad un dato *Cliente*, invece, definiamo a livello globale il *RecordSet RstCliente*.

```
Public RstCliente As ADODB.Recordset
```

che carichiamo attraverso la seguente procedura:

```
Public Sub caricacliente(cliente As Integer)
 Dim strcnn As String
 strcnn = "Provider=microsoft.jet.oledb.4.0;" _
 & "Data Source=" + App.Path + "\ordini.mdb; "
 Set RstCliente = New ADODB.Recordset
 RstCliente.CursorType = adOpenKeyset
 RstCliente.LockType = adLockOptimistic
 RstCliente.Open "Select * from ordini where _
 clienteid=" & cliente, strcnn, , , adCmdText
End Sub
```

L'argomento *cliente*, della *caricacliente*, è impostato sulla scheda *Imposta* (*txtclienteid*). Il codice per creare un nuovo *Cliente* o per cancellarlo lo inseriamo nei pulsanti *Nuovo* e *Cancella*:

```
Private Sub Nuovo_Click()
 Adodc1.Recordset.AddNew
 Txtcitta.SetFocus
 ` Clienteid è un contatore, il focus
 `l'impostiamo su txtcittà
End Sub
Private Sub Cancella_Click()
 Adodc1.Recordset.Delete
 Adodc1.Recordset.Requery
End Sub
```

CARICARE IL FORM

In questo paragrafo inseriamo il codice (da inserire nella *Form_ Load*) che serve per impostare le colonne del *ListView* e per caricare i dati del primo *Cliente* (con la *CaricaControlloAdoCliente*).

```
Private Sub Form_Load()
 ListView1.ColumnHeaders. _
                  Add , , "ID", 400, lvwColumnLeft
 ListView1.ColumnHeaders. _ Add , , "Prezzo", , _
                                  lvwColumnCenter
 ListView1.ColumnHeaders. _ Add , , "NPezzi", , _
 lvwColumnCenter
 ListView1.ColumnHeaders. _
 Add , , "Note", ListView1.Width / 2, lvwColumnCenter
 ListView1.View = lvwReport
 Picture1(1).Visible =False
```

```
'imposta la scheda che non si deve vedere
CaricaControlloAdoCliente
End Sub
```

GESTIONE DEL TABSTRIP

Le schede del *TabStrip* (realizzate con un array di picturebox) possono essere gestite con il seguente codice:

```
Private Sub TabStrip1_Click()
  If TabStrip1.SelectedItem.Index <> 1 Then
                                   Picture1(0).Visible = False
  If TabStrip1.SelectedItem.Index <> 2 Then
                                   Picture1(1).Visible = False
  Picture1(TabStrip1.SelectedItem.Index - 1).Visible = True
End Sub
```

Oppure utilizzando un oggetto *UpDown* nel quale inseriamo il seguente codice:

```
Private Sub UpDown1_DownClick()
 TabStrip1.Tabs(1).Selected = True
 TabStrip1_Click
End Sub
Private Sub UpDown1_UpClick()
 TabStrip1.Tabs(2).Selected = True
 TabStrip1_Click
End Sub
```

RICERCA ORDINI

In conclusione presentiamo il codice per caricare gli ordini del cliente specificato in *TxtCodice*, cioè la procedura *Trova_Click*:

```
Private Sub Trova_Click()
 If txtcodice <> 0 Then
  caricalista (txtcodice)
  TabStrip1.Tabs(2).Selected = True
  TabStrip1_Click
 End If
End Sub
```

La procedura precedente dopo aver caricato il *ListView,* attraverso la *caricalista,* seleziona la scheda *Ricerca.* La *caricalista* è la seguente:

```
Public Sub caricalista(codice As Integer)
 Dim itmX As ListItem
 ListView1.ListItems.Clear
 caricacliente (codice)
 While Not RstCliente.EOF
   Set itmX = ListView1.ListItems. _
   Add(, , CStr(RstCliente.Fields.Item(0)))
```

```
   itmX.SubItems(1) = RstCliente.Fields.Item(3)
   itmX.SubItems(2) = RstCliente.Fields.Item(4)
   itmX.SubItems(3) = RstCliente.Fields.Item(2)
   RstCliente.MoveNext
 Wend
End Sub
```

Essa ripulisce il ListView, carica *RstCliente* con i dati degli ordini del cliente (con la *caricacliente*) e riempie il *ListView.*

DISTRIBUIRE IL FILE MANIFEST

In questo paragrafo descriveremo come includere il file *Manifest*, direttamente, in un progetto dopo averlo trasformato, utilizzando il tool *RC.exe*, in un file di risorsa (cioè in un file *.RES*). Creiamo un file *.rc* e lanciamo il compilatore *RS.EXE* con un file *.BAT*. Allora inserire i comandi di sotto in un file nominato *progetto1.rc* (file testo con estensione .rc) da salvare nella stessa directory del file *Progetto1 .exe.manifest*.

Fig. 4: L'applicazione in stile Windows 95

```
#define CREATEPROCESS_MANIFEST_RESOURCE_ID 1
#define RT_MANIFEST 24
CREATEPROCESS_MANIFEST_RESOURCE_ID
                    RT_MANIFEST   "Progetto1.exe.manifest"
```

Con i comandi precedenti si specifica che il file *Progetto1.exe.manifest* deve essere classificato come risorsa tipo *RT_MANIFEST (24)* con id *CREATEPROCESS_MANIFEST_RESOURCE_ID (1).*
Poi create il file *BAT,* per esempio *lanciaRC.bat*, con la seguente riga di comando:

```
C:\Programmi\Microsoft Visual
                        Studio\VB98\Wizards\RC.EXE" /fo
Progetto1.res Progetto1.rc
```

Anche *lanciaRC.bat* deve essere salvato nella stessa directory del file *Progetto1.exe.manifest.* Facciamo notare uno strano errore: il file *EXE,* compilato dopo aver incluso il file *Progetto1.res* (prodotto con il compilatore *RC.EXE*), si avvia soltanto se le dimensioni, in byte, del file *progetto1.exe.manifest* sono multipli di 4 (se *progetto1.exe.manifest* è di 595 byte per renderlo multiplo di 4 potete inserire un carattere spazio).

Massimo Autiero

INITCOMMON CONTROLS

La *InitCommonControls* viene usata per controllare lo stato di caricamento della libreria *COMCTL32.DLL*. A volte, però, se l'applicazione include le Windows *Common Controls* di Visual Basic, non è necessario usare la *InitCommonControls* (verificatelo con le vostre applicazioni!). Questo, per esempio, succede nel caso *VB6.exe* che usa lo stile XP solo grazie al file *vb6.exe.manifest.*

XML per progettare le interfacce grafiche

Interfacce vettoriali in XAML

Chi ha avuto la fortuna di provare Longhorn, avrà sicuramente notato un sostanziale cambiamento anche nell'interfaccia grafica Non tutti sanno che il tutto è stato realizzato tramite XML…

Longhorn sarà la futura versione di Windows e porterà con sé non poche novità, legate, soprattutto, all'interazione con XML.
Lo stesso motore grafico (Avalon), che darà vita alla nuova interfaccia utente, è stato interamente progettato per interagire con un linguaggio direttamente derivato da XML e denominato XAML, acronimo di *Microsoft Extensible Application Markup Language*.
L'intera grafica di un'applicazione XAML è "curata" da un insieme di tag XML che, interagendo con codice .NET di alto livello (C#, VB.NET, C++.NET,C#), danno vita all'interfaccia grafica.
Si, è vero, in molti già si chiederanno: ioProgrammo mi spiega una tecnologia che potremo utilizzare solo nel 2006! Niente paura, per venire in contro ai tanti curiosi, c'è stato chi ha pensato bene di realizzare un tool che consente di utilizzare da subito la nuova tecnologia.
Lo sviluppatore Paul Colton ha messo a disposizione un software xamlon (*www.xamlon.com*) che consta di due strumenti:

Fig. 1: ***Semplice ed efficace l'interfaccia di XamlPad***

Visual Designer for Visual Studio .NET 2003 e *XamlPad*. Il primo mette a disposizione un plug-in per Visual Studio .NET 2003, che permette di salvare l'interfaccia grafica di un'applicazione in linguaggio XAML; il secondo implementa, invece, una sorta di Notepad per scrivere e testare script XAML; in aggiunta consente finanche di convertire file SVG e C# in linguaggio XAML nativo (Fig.1).
L'installazione del pacchetto prevede che sul proprio sistema operativo sia correttamente installata la versione 1.1 del .NET Framework; quest'ultimo downloadabile direttamente dal sito Microsoft nell'area download.

LA SINTASSI XAML

La sintassi di uno script XAML è molto simile a quella di SMIL, il linguaggio a marcatori di tag, sviluppato come estensione XML, per l'integrazione di oggetti multimediali nei siti web.
Tipicamente, uno script XAML si presenta come un classico file XML con, in aggiunta, tutti i riferimenti per la creazione e gestione di una interfaccia grafica in stile Windows:

```
<?Mapping XmlNamespace="wf" ClrNamespace=
                              "System.Windows.Forms"
Assembly="System.Windows.Forms" ?>
<wf:Form Name="Form1" xmlns=
              "http://schemas.microsoft.com/2003/xaml"
xmlns:def="Definition" xmlns:wf="wf" Text=
                           "La prima applicazione XAML"
Visible="true">
    <wf:Button BackColor="Red">Ciao
mondo!</wf:Button>
</wf:Form>
```

Per “eseguire” il codice: caricare l’applicazione XamlPad, trascrivere il codice e premere il tasto *F5* per avviare il motore di rendering. L’output generato sarà molto fedele a quanto proposto in Fig.2

Fig. 2: ***La nostra prima applicazione xaml: l’intramontabile Ciao Mondo!***

Un file XAML viene interpretato e compilato in fase di runtime; nel particolare, il compilatore XAML genera, per ogni file XAML, una specifica classe .NET. Le classi conterranno il codice per generare gli oggetti definiti nello script XAML. Gli elementi xml, e i relativi attributi, sono quindi “mappati” come proprietà di codice .NET. Per esempio l’elemento:

```
<wf:Button BackColor="Red">Ciao mondo”
                                    </wf:Button>
```

Sarà mappato in codice C# nel modo seguente:

```
Button_3_.Backcolor =
    new MSAvalon.Windows.Media.SolidColorBrush(
            MSAvalon.Windows.Media.Color.FromARGB(
                                    255, 255, 0, 0)
```

Notate come le classi siano direttamente invocate dal nuovo oggetto *MSAvalon* che, come già detto in precedenza, rappresenta il cuore pulsante della nuova interfaccia grafica di Longhorn.

NOTA

AVALON

Avalon è il nuovo motore grafico della prossima release di Windows denominata Longhorn; nello specifico si tratta di un nuovo set di API che "aiutano" lo sviluppatore nella costruzione di applicazioni evolute dal punto di vista multimediale. Con Avalon Microsoft dovrebbe ridurre il numero di API da 70 a 8 mila facendo leva sulle avanzate caratteristiche delle nuove schede grafiche in commercio.

NON SOLO 2D

XAML essendo nato per progettare interfacce grafiche, consente di invocare primitive sia bidimensionali che tridimensionali.
Chi ha avuto modo di lavorare con SVG, lo standard xml per generare grafica 2D, troverà in XAML molte similitudini, per esempio la primitiva *path:*

```
<Path
xmlns="http://schemas.microsoft.com/2003/xaml"
    Fill="LightGreen" Stroke="DarkGreen"
    StrokeThickness="2">
    <Path.Data>
        <GeometryCollection>
            <EllipseGeometry Center="65,65"
                RadiusX="55" RadiusY="55"/>
            <EllipseGeometry Center="135,65"
                RadiusX="55" RadiusY="55"/>
            <EllipseGeometry Center="65,135"
                RadiusX="55" RadiusY="55"/>
            <EllipseGeometry Center="135,135"
                RadiusX="55" RadiusY="55"/>
        </GeometryCollection>
    </Path.Data>
</Path>
```

Tale primitiva, in SVG utilizzata in modo lievemente diverso, consente di definire un percorso grafico e di associare a questo diversi attributi. Nel codice d’esempio, *Path* è adoperata per creare una “collezione” di oggetti geometrici formata da ellissi. Il colore di riempimento delle ellissi è settato a verde chiaro: *Fill="LightGreen"* mentre il colore che delinea il contorno delle stesse è settato a verde scuro: *Stroke="DarkGreen";* gli attributi *RadiusX, RadiusY* e *Center,* impostano raggio e centro dell’ellisse.
Il grafico generato dal codice è quello rappresentato in Fig.3

Fig. 3: ***Poche righe di codice XAML per generare grafica 2D!***

XAML include anche elementi di supporto per la creazione di grafica 3D. Allo stadio di sviluppo attuale, al momento in cui scriviamo è stata rilasciata la Beta 5, sono definiti il

supporto per la gestione dei mesh in modalità wireframe e solid e la gestione delle luci d'ambiente e di quelle direzionali.
In particolare, sono implementati elementi per il posizionamento di una videocamera virtuale, per il setting delle luci d'ambiente, per la scelta dei materiali, ecc.
Nell'esempio che segue viene mostrato come realizzare due cubi rispettivamente di colore giallo (*<BrushMaterial Brush="yellow"/>*) e blu (*<BrushMaterial Brush= "blue"/>*. L'operazione è resa possibile grazie all'elemento *Mesh3D* che consente di definire tutti i punti che compongono la figura geometrica tridimensionale, così da rendere possibile qualunque tipo di rappresentazione.

```
<?xml version="1.0"?>
<Canvas
xmlns="http://schemas.microsoft.com/2004/xaml">
 <ViewPort3D ShadingMode="Flat">
  <ViewPort3D.Camera>
   <PerspectiveCamera Position="-50,50,100"
                 LookAtPoint="0,0,0" Up="0,1,1"
NearPlaneDistance="0"
FarPlaneDistance="20" FieldOfView="40"/>
  </ViewPort3D.Camera>
  <ViewPort3D.Models>
   <Model3DCollection>
    <AmbientLight Color="#404040"/>
    <DirectionalLight Color="#C0C0C0"
                 Direction="0.5,-0.25,-1"/>
    <MeshPrimitive3D>
     <MeshPrimitive3D.Material>
      <BrushMaterial Brush="yellow"/>
     </MeshPrimitive3D.Material>
     <MeshPrimitive3D.Mesh>
      <Mesh3D Positions="-100,-100,-100 0,-100,
     -100 0,0,-100 -100,0,-100 -100,-100,0 0,-
100,0 0,0,0 -100,0,0" TriangleIndices="0,1,2 0,2,3
                  1,5,6 1,6,2 5,4,7 5,7,6
           4,0,3 4,3,7 3,2,6 3,6,7 4,5,1 4,1,0"/>
     </MeshPrimitive3D.Mesh>
    </MeshPrimitive3D>
    <MeshPrimitive3D>
     <MeshPrimitive3D.Material>
      <BrushMaterial Brush="blue"/>
     </MeshPrimitive3D.Material>
     <MeshPrimitive3D.Mesh>
      <Mesh3D Positions="0,0,0 100,0,0 100,100,0
        0,100,0 0,0,100 100,0,100 100,100,100
0,100,100" TriangleIndices="0,1,2 0,2,3 1,5,6 1,6,2
  5,4,7 5,7,6 4,0,3 4,3,7 3,2,6 3,6,7 4,5,1 4,1,0"/>
     </MeshPrimitive3D.Mesh>
    </MeshPrimitive3D>
   </Model3DCollection>
  </ViewPort3D.Models>
 </ViewPort3D>
</Canvas>
```

Fig. 4: ***XAML permette anche la rappresentazione di oggetti tridimensionali***

DISEGNARE L'INTERFACCIA E SALVARLA IN XAML

La grande forza di XAML consiste nel dare la possibilità all'utente di disegnare l'interfaccia grafica di un'applicazione, utilizzando una qualunque applicazione di grafica vettoriale; per esempio, possiamo pensare di disegnare il tutto adoperando Adobe Illustrator e, successivamente, trasformare il disegno in linguaggio XAML. Procediamo con un esempio che traduca in pratica quanto appena asserito.
Con Adobe Illustrator decidiamo di ideare un logo.
Per poter trasformare il file in formato XAML è necessario salvare il documento Illustrator in formato SVG (menu *File->Salva con nome* e scegliere come formato di salvataggio SVG) Dalla maschera selezioniamo il bottone *Avanzate* e dalle *Proprietà CSS* scegliamo l'opzione *Presentazione Attributi*, deselezionando tutte le altre opzioni; quindi procediamo con il salvataggio.
A procedura ultimata avviamo l'applicazione *XamlPad* e dal menu *File* selezioniamo la voce *Import* specificando il file *SVG* poco prima salvato. Ecco il codice *XAML* generato, frutto dell'importazione:

```
<Canvas def:def="Definition" Width="265" Height=
              "171.19" xmlns:def="Definition">
 <Rectangle RectangleLeft="0" RectangleTop=
```

NOTA

SCALABLE VECTOR GRAPHICS

SVG acronimo di Scalable Vector Graphics è un linguaggio di grafica vettoriale 2D che basa le sue fondamenta su XML; SVG è stato sviluppato e approvato dal consorzio W3C e la sua ultima release è targata 1.1. Attualmente lo standard è stato ulteriormente evoluto per consentirne l'utilizzo anche agli sviluppatori del mondo mobile.

```
                         "11.432" RectangleWidth="165"
RectangleHeight="147" Fill="#FF0000" />

<Polygon Fill="#FF0000" Points="265,90.432
                       209.349,112.714 196.154,171.19
157.766,125.148 98.074,130.669 130,79.932
                         106.303,24.868 164.423,39.553
209.469,0 213.463,59.813" />
</Canvas>
```

Procedendo con il rendering (tasto *F5*) otterremo il medesimo grafico progettato con Adobe Illustrator (Fig.5). Al momento in cui scriviamo la versione beta di Xamlon presenta alcuni bug che si manifestano nell'import di file nei quali sono mappati font di sistema.

Fig. 5: Logo elaborato con Adobe Illustrator

LA GESTIONE DEGLI EVENTI

XAML non consente di gestire direttamente gli eventi legati agli oggetti istanziati, per esempio non è possibile definire una risposta all'evento *click* di un bottone utilizzando elementi del linguaggio XAML; per ovviare a ciò, XAML si integra con codice .NET ad alto livello, tipicamente Visual Basic .NET e/o C#. L'esempio che segue mostra come creare un pulsante e associare del codice C# che ne intepreti l'evento *click*; anche in questo caso, così come accade in xml, si fa uso della direttiva *CDATA* per impartire al compilatore istruzioni "estranee" allo standard:

```
<Canvas ID="root"
xmlns="http://schemas.microsoft.com/2003/xaml"
xmlns:def="Definition">
  <Button ID="button1" Click="Clic_Bottone">
                                Cliccami!</Button>
    <def:Code>
      <![CDATA void Clic_Bottone(
                   object target, ClickEventArgs args)
      {
           button1.content="Ciao Mondo!";
```

```
        }
        ]]>
    </def:Code>
</Canvas>
```

IL PLUG-IN PER VS 2003

Come già annunciato in precedenza, il tool Xamlon installa un plug-in per VS 2003 che consente di utilizzare l'IDE del pacchetto di sviluppo per creare interfacce XAML, e linkare queste con codice .NET; è così possibile creare applicazioni .NET la cui interfaccia grafica risieda in un file XAML, modificabile a proprio piacimento, e la parte eseguibile risieda in un ulteriore file, tipicamente .EXE.
Per provare praticamente il plug-in, basta avviare VS .NET 2003 e creare un nuovo progetto. Dal menu file selezionare le voci *Nuovo->Progetto* e dalla cartella *Progetti* di Visual C# optare per la creazione di un nuovo progetto *XamlonProjects*, quindi scegliere *Xamlon Application*. A questo punto l'ambiente è operativo.
Proviamo a creare una semplice applicazione che, alla pressione di un generico button, cambi la caption dello stesso con la stringa *"Ciao Mondo!"*. Da *Esplora soluzioni* di VS .NET 2003 clicchiamo sul *Form1.xaml* per accedere alla finestra principale dell'applicazione, quindi posizioniamo il button al centro della finestra; da notare come l'IDE incorpori due pulsanti distinti: *Design* e *XAML*, rispettivamente per "switchare" dalla fase di "disegno" RAD dell'applicazione alla fase scrittura manuale del codice.

Fig. 6: I due file generati da VS 2003 nella creazione di un'applicazione XAML

A questo punto si procede, così come in ogni altra applicazione .NET, associando all'evento di pressione del pulsante l'azione di assegnazione della nuova caption *"Ciao mondo!"*:

NOTA

LONGHORN

Solo di recente Microsoft ha dichiarato che la prossima versione di Windows non vedrà luce prima del 2006. Alcune delle tecnologie che prima Microsoft aveva deciso di adoperare per Longhorn saranno posticipate (una su tutte WinFS) mentre altre, vedi Avalon e Indigo, presenti in Longhorn, saranno anche rilasciate come upgrade alla attuali versioni di Windows XP.

```
using System;
using System.Windows.Forms;
using Xamlon.Windows;
using Xamlon.Windows.Controls;
namespace XamlonApplication2
{
    public class Form1 : _Form1
   {
       public void Button1_Click(Object sender, EventArgs e)
       {
             Button1.Text="Ciao Mondo!";
        }
   }
}
```

Si prosegue generando la soluzione.
In Fig.6 è presente uno screenshot che mostra i file generati: un file eseguibile: *XamlonApplication2.exe* ed un file XAML: *form1 .xaml*. Il codice contenuto nel file *form1 .xaml* sarà del tipo:

```
<?Mapping XmlNamespace="wf" ClrNamespace=
                    "System.Windows.Forms" Assembly=
                              "System.Windows.Forms"?>
<?Mapping XmlNamespace="events" ClrNamespace=
                         "XamlonApplication2" Assembly=
                                   "XamlonApplication2"?>
<wf:Form ID="Form1" Width="328" Top="13"
          Text="Form1" Left="13" IsAccessible="False"
     Height= "200" Capture="False" Bounds="13, 13,
                                             328, 200"
  TabIndex="0" DialogResult="None" DesktopLocation=
                        "13, 13" DesktopBounds="13, 13,
328, 200" ClientSize="320, 173" AllowTransparency=
        "False" xmlns="http://schemas.microsoft.com
 /2005/xaml/" xmlns:def="Definition" xmlns:wf="wf"
              def:Class="XamlonApplication2.Form1">
 <wf:Form.Controls>
    <wf:Button ID="Button1" Width="256" Top="48"
            Text="Button1" TabIndex="0" Size="256,
56" Location="32, 48" Left="32" IsAccessible="False"
  Height="56" ClientSize="256, 56" Capture="False"
                               Bounds="32, 48, 256, 56"
Click="Button1_Click" />
  </wf:Form.Controls>
</wf:Form>
```

Se proviamo a variare qualche parametro del file *XAML*, per esempio aggiungendo l'attributo *BackColor="Red"*:

```
...
...
...
wf:Button ID="Button1" BackColor="Red"
 Width="256" Top="48" Text="Button1" TabIndex="0"
Size="256, 56" Location="32, 48" Left="32"
                        IsAccessible="False" Height="56"
ClientSize="256, 56" Capture="False" Bounds="32,
                    48, 256, 56" Click="Button1_Click" />
...
...
```

noteremo che l'applicazione *XamlApplication2*, senza dover ricompilare il tutto, interpreterà il codice XAML e modificherà la propria interfaccia grafica in funzione dello stesso. Semplice e geniale.

***Fig. 7:** L'applicazione XamlApplication2 prima e dopo l'aggiunta dell'attributo BackColor="Red" al codice di form1.xaml*

Inutile sottolineare la potenzialità di questo strumento: pensate alla possibilità di variare, a proprio piacimento, l'interfaccia grafica di una qualsiasi applicazione Windows, magari utilizzando il programma di grafica vettoriale preferito.

CONCLUSIONI

XAML rappresenta solo una delle tante nuove tecnologie che dovrebbero far parte del nuovo SO di casa Microsoft; certo che Microsoft, con XAML, propone una soluzione tecnologica che sicuramente cambierà lo sviluppo delle applicazioni Windows.
Grazie a Xamlon, nell'attesa di avere delle specifiche complete, e soprattutto in attesa della nuova release di Longhorn, possiamo adoperarci per far si che le nostre applicazioni siano già in un certo senso Longhorn enabled.

Estrazione di dati da un file binario con C#

iPod: estrazione dei brani con C#

L'iPod è forse il lettore audio più diffuso al mondo. È facile inserirvi la nostra musica, ma non è così immediato copiare i file dall'iPod al nostro PC. Realizziamo una classe C# per estrarre dati e musica

L'iPod propriamente detto, di cui è uscita da poco la quarta generazione, è un juke box di grande capienza (fino a 40Gb), mentre l'iPod mini, il bellissimo e piccolissimo lettore colorato uscito qualche mese fa è un gioiellino da 4Gb le cui dimensioni ridottissime e il cui design accattivante hanno fatto impazzire mezzo mondo (compreso chi scrive). L'iPod legge i principali formati di compressione audio, eccezion fatta per WMA. In particolare legge MP3, AAC (il codec proprietario di Apple, in cui possono essere immesse le informazioni di protezione dalla copia) e il nuovissimo Apple LossLess codec. Oltre ai brani musicali, è possibile registrare appuntamenti e brani di testo. L'iPod può essere connesso al PC tramite USB e FireWire e può anche essere visto dal PC come un disco ed essere quindi utilizzato per scrivere e leggere file.

Esiste un problema: non è possibile in modo diretto "estrarre" e copiare sul proprio hard disk i brani voluti. Per esempio, non c'è una cartella che contiene tutti i brani di un dato CD o di un autore: tali brani saranno sparsi per differenti cartelle, insieme a brani appartenenti ad album e autori differenti. L'ottimo software fornito da Apple per interagire con l'iPod, ovvero iTunes, permette di inviare all'iPod brani musicali nei formati supportati, sia direttamente che dopo averli convertiti. Non permette però di estrarre i brani e copiarli sul PC.
Il nostro scopo sarà semplicemente quello di individuare il path di ogni brano presente nell'iPod, per poter poi operare l'estrazione, che sarà semplicemente la copia di un file dal "disco" iPod al disco rigido del nostro PC. Scriveremo a tale scopo una classe C#.

REQUISITI

Conoscenze richieste
Basi di programmazione ad oggetti, basi di C#

Software
.NET framework 1.1, Visual Studio 2003 utile ma non obbligatorio.

Impegno

Tempo di realizzazione

UN GIOIELLINO E UN DATABASE

A differenza di molti altri lettori mp3 (in particolar modo quelli a stato solido) il modello di accesso ai brani dell'iPod non è basato su cartelle, ma su un database: ovvero all'interno dell'unità c'è un file che contiene informazioni come autore, titolo, album, genere per ogni brano presente nel dispositivo. I file veri e propri risiedono in cartelle, e i percorsi di tali cartelle sono memorizzate nel database. Questa struttura a database ha un chiaro vantaggio, ovvero quello di permettere una navigazione dei brani secondo differenti "viste":

- per playlist
- per autore
- per album
- per genere
- per titolo

ALL'INTERNO DEL DB

Per capire come estrarre i nostri file mp3, andiamo a vedere come è fatto l'albero delle cartelle del disco contenuto nell'iPod.

```
----Contacts
----Calendars
----Notes
_   ----iPod Agent
_       +---Library
----iPod_Control
   ---Device
   ----Library
   ----Music
          ---F01
          ---F02
          ---F03
          ---F04
           ...
```

Il file che contiene le informazioni volute è *iTunesDB,* contenuto nella cartella *iPod_Control.* I nostri file musicali sono invece contenuti nelle varie cartelle *iPod_Control\Music\FXX,* con *XX* che varia. Visto che vogliamo estrarre i dati, leggeremo soltanto e non scriveremo sul db. L'inserimento dei file, infatti, è procedura ben più complessa e rischiosa: se il DB risultasse corrotto, il nostro iPod diverrebbe inutilizzabile. Per sicurezza, anche nel nostro caso di lettura consiglio di farsi una copia sul PC del file di database, e lavorare lì almeno fino a che non siamo convinti che il nostro codice sia corretto. Il file *iTunesDB* è un file binario, caratterizzato da una sequenza di sezioni delimitate da intestazione di quattro caratteri. Ogni intestazione è seguita dalle informazioni riguardanti la sezione e a volte le sezioni successive. Se questa è la struttura "fisica" del file, la struttura "logica" del database può essere vista come un albero:

MHBD	ce n'è solo uno, marca l'inizio del db
MHSD di tipo 1	inizio delle informazion sui brani
MHLT	
MHIT	inizio di un brano e info sul brano
MHOD	dato stringa per il brano
MHOD	dato stringa per il brano
...	continuano i dati fino all prossimo brano
MHIT	inizio di un brano, ecc
...	
MHSD di tipo 2	inizio delle informazioni sulle playlist
MHLP	contiene il numero di playlist nel db
MHYP	inizio di una playlist e info sulla stessa
MHOD di tipo 100	info sulla playlist
MHOD	info sul nome della playlist
MHIP	brano presente sulla playlist
MHOD di tipo 100	Ordine del brano sulla playlist
MHIP	...
MHOD di tipo 100 ...	
...	
MHYP	
...	

Vedremo quindi come effettuare il parsing di un file binario con C# e come utilizzare un *DataSet* tipizzato come repository naturale dei dati estratti.

RIDAMMI I MIEI FILE!

Per la lettura del file utilizzeremo un oggetto di classe *FileStream* e uno di classe *BinaryReader,* chiaramente dopo esserci assicurati di aver impostato la modalità *Hard Disk* per l'iPod. Questa caratteristica, impostabile facilmente in iTunes, ci permette di leggere il contenuto dell'iPod in maniera del tutto analoga a quanto facciamo con una qualsiasi unità disco. In queste pagine, per semplicità e brevità, estrarremo da ogni sezione soltanto le informazioni che ci servono per il nostro scopo. L'algoritmo che utilizzeremo per l'estrazione è estremamente semplice:

- inizializziamo un oggetto di tipo *BinaryReader.* Andiamo a leggere nel file, per ogni intestazione che troviamo andremo a leggere la sezione adeguata, e sposteremo il puntatore del *BinaryReader* del numero di byte necessario per arrivare all'inizio della prossima sezione. In dettaglio:

    1. estraiamo e saltiamo la sezione di testa, segnata dal intestazione *"mhdb".*
    2. estraiamo e saltiamo la sezione *MHSD* di tipo 1 e la *MHLT*. Siamo ora nella sezione dei brani.
    3. per ogni brano, troviamo una sezione *MHIT*.

- Creiamo una riga *BranoRow* nella *DataTable Brano* del nostro *DataSet* e immettiamo i dati.
- Estraiamo dalla sezione *MHIT* il numero *N* delle sezioni *MHOD* presenti
- Estraiamo per ogni *MHOD* la stringa corrispondente e la assegniamo al giusto campo della riga *BranoRow*
- Quando abbiamo estratto *N MHOD,* aggiungiamo la riga alla *DataTable*
- quando troveremo *MHSD* di tipo 2 saranno finiti i brani e cominceranno le playlist.
- saltiamo *MHLP*
- per ogni playlist analizziamo il *MHYP* e utilizziamo *MHOD, MHOD* e *MHIP* per estrarre le informazioni che ci servono.

in realtà le informazioni sulle playlist non ci servono realmente per l'estrazione: sarà però comodo averle sotto mano per organizzare la visualizzazione dei brani nella nostra applicazione.

UN'ESTRAZIONE DI CLASSE

Il codice dell'estrazione sarà contenuto in una classe, *iTunesDBExtractor*. Abbiamo detto inoltre che utilizzeremo un *DataSet* tipizzato. Non solo, lo creeremo anche con le relazioni fra le tabelle, in modo da poter più agevolmente "navigare" attraverso i dati. Le *DataTable* di cui avremo bisogno sono essenzialmente tre, che realizzano una classica relazione molti-a-molti:

1. **Brano:** conterrà i dati dei brani contenuti nell'iPod
2. **Playlist:** conterrà dati identificativi delle varie playlist, compreso il tipo. Se una Playlist è di tipo 1, allora è la playlist principale, con tutti i brani.

Altrimenti è di tipo 0 ed è una playlist ordinaria.

3. **Relazionebranoplaylist:** contiene chiavi esterne alle altre due tabelle e permette di associare ogni brano a più paylist.

Fig. 1: ***Il DataSet creato in VisualStudio 2003. Il DataSet realizza una classe che chiameremo ipodDS***

All'inizio della nostra classe dobbiamo creare alcuni membri privati, una proprietà che ci aiuterà ad impostare il percorso al file di DB ed i costruttori:

```
public class iTunesDBExtractor
{  //impostazione sezione DataSet
   ipodDS _DS;
   //la DataRow che conterrà il brano sotto estrazione
   ipodDS.BranoRow  _branoCorrenteRow;
   //le DataRow che conterranno i dati della playlist
                                              sotto estrazione
   ipodDS.PlayListRow _playlistCorrenteRow;
   ipodDS.RelazionePlayListBranoRow
                                    _playlistBranoCorrenteRow;
   //gli oggetti che serviranno alla lettura
   BinaryReader _bReader;
   FileStream _fileStr;
   //privato per la prperty
   string _percorsoDB="";
   //flags di stato
   bool _sezioneBrani = false;
   bool _inLetturaPlaylist = false;
   int _playlistIndex = 0;
   //numero di stringhe contenute nel brano corrente
   int _numerodistringhe=0;
   /// <summary>
   /// proprietà che imposta il path al file ITunesDB
   /// </summary>
   public string PercorsoDB
   {  set
      { _percorsoDB=value; }
        get
        { return _percorsoDB; }
     }
     /// <summary>
     /// costruttore di default
     /// </summary>
     public iTunesDBExtractor()
     {}
     /// <summary>
     /// costruttore che permette l'impostazione del path
     /// </summary>
     /// <param name="PercorsoDB"></param>
     public iTunesDBExtractor(string PercorsoDB)
     { _percorsoDB=PercorsoDB; }
```

Avremo ora bisogno di tre funzioni helper. Le prime due ci aiuteranno ad estrarre stringhe ASCII e Unicode dai byte letti dal nostro *BinaryReader*, l'altra ci servirà a saltare sezioni di cui non ci interessa il contenuto. Il funzionamento di quest'ultima è semplice se si considera che generalmente i primi quattro byte di una sezione sono l'intestazione (*mhdb*, *mhod* ecc) mentre i secondi quattro indicano la lunghezza della sezione:

```
/// <summary>
/// converte un buffer di byte in una stringa ASCII
/// </summary>
/// <param name="buffer"></param>
/// <returns></returns>
private string LeggiStringaA(byte[] buffer)
{ return System.Text.Encoding.ASCII.GetString(buffer);
}
/// <summary>
/// converte un buffer di byte in una stringa Unicode
/// </summary>
/// <param name="buffer"></param>
/// <returns></returns>
private string LeggiStringaU(byte[] buffer)
{ return System.Text.Encoding.Unicode.GetString(buffer);}
/// <summary>
/// passa alla prossima sezione per le sezioni di cui non
/// interessa il contenuto
/// </summary>
private void SaltaSezione()
{  //i primi quattro bytes ci dicono la lunghezza del record
    int Lunghezza = _bReader.ReadInt32();
    //arriviamo alla fine dell'intestazione
    //che si trova a Lunghezza -8 (intestazione+campo
                                          lunghezza) byte
   _bReader.ReadBytes(Lunghezza - 8);
}
```

Siamo arrivati al cuore della classe, ovvero al metodo che estrarrà i dati dal DB e che popolerà il DataSet:

```
/// <summary>
/// metodo publico principale che estrae il dataset con
                                    le informazioni volute
/// </summary>
/// <returns></returns>
public ipodDS ExtractDS()
{  try
   { if (PercorsoDB=="") throw new Exception(
                             "percorso al DB non inizializzato");
```

Intestazione di Sezione: **ogni sezione inizia con quattro byte riconoscibili.**

iTunesDB**: il file contenuto nell'iPod ove il lettore immagazzina i dati sui brani che contiene, allo scopo di fornire differenti viste all'utente.**

MP3, AAC**: alcuni dei formati di compressione audio supportati da iPod**

Playlist**: una lista di brani**

Sezione**: il file di DB è diviso in sezioni contenenti differenti informazioni**

```
    //creo un file stream
    _fileStr = new FileStream(_percorsoDB,
                                        FileMode.Open);
    //apro il BinaryReader su questo filestream
    _bReader = new BinaryReader(_fileStr);
    //imposto il Dataset.
    //per evitare problemi e migliorare le prestazioni
    //imposto a false il controllo dei constraints
    _DS= new ipodDS();
    _DS.EnforceConstraints=false;
    //leggo la prima intestazione
    string intestazione = LeggiStringaA(
                               _bReader.ReadBytes(4));
    //entro nel ciclo principale
    while (intestazione.Length>0)
    { //a seconda dell'intestazione leggo la sezione
                                                voluta.
        switch (intestazione)
        { ... }
      if (_inLetturaPlaylist)
      {  _DS.PlayList.Rows.Add(_playlistCorrenteRow);
         _inLetturaPlaylist = false; }
      //rimettiamo i constraints nel dataset
      _DS.EnforceConstraints=true;
      return _DS; }
    catch (Exception ex)
   { throw ex; }
    finally
   { if (_bReader!=null) _bReader.Close();
     if (_fileStr!=null) _fileStr.Close();}
}
```

Il funzionamento del metodo è il seguente: una volta creati gli oggetti *FileStream* e *BinaryReader* che realizzeranno la lettura del file, viene creata l'istanza del *DataSet* che verrà restituita. Dato che l'ordine di inserimento delle righe nelle *DataTable* non rispetta le relazioni del *DataSet*, vengono disabilitati temporaneamente i controlli di correttezza tramite l'istruzione:

```
_DS.EnforceConstraints=false;
```

Viene letta la prima intestazione e poi si dà inizio ad un classico ciclo *while* in cui è presente un'istruzione di selezione che esegue differenti operazioni a seconda dell'intestazione letta e dello stato della lettura. Le funzioni chiamate estraggono i dati dalle varie sezioni e verranno illustrate di seguito. In particolare vediamo che per alcune sezioni viene invocata la funzione *SaltaSezione* che permette di passare velocemente alla prossima sezione senza estrarre dati dalla sezione interessata. Chi volesse invece estrarre tali informazioni può consultare i vari documenti riportati nei box laterali ed operare l'estrazione con le tecniche utilizzate in questo articolo. Alla fine del ciclo *while* viene letta la prossima intestazione e così via fino alla fine. Una volta usciti dal ciclo, viene aggiunta l'ultima riga alla tabella *PlayList* e vengono ripristinati i controlli sul *DataSet* che viene restituito al chiamante dopo la chiusura degli oggetti di lettura. Ma andiamo a vedere le funzioni chiamate per effettuare l'estrazione vera e propria.

```
/// <summary>
/// estrazione dei dati della sezione Mhit, per un brano
/// </summary>
private void EstraiSezioneMhit()
{  //alcune informazioni vengono messe nella riga
   //altre vengono estratte per usi futuri
   int LunghezzaSezione = _bReader.ReadInt32();
   int LunghezzaSezioneEFigli = _bReader.ReadInt32();
   ...
}
```

La funzione *EstraiSezioneMhit* viene chiamata dal ciclo principale dopo aver incontrato un'intestazione *"mhit"* e dopo aver creato un oggetto di tipo *ipodDS.BranoRow*. Tale oggetto, *_branoCorrenteRow*, verrà utilizzato per inserire i dati del brano corrente e poi verrà aggiunto alla *DataTable_DS .Brano*. Nel codice della funzione infatti si vede come alcuni dei valori letti vengono inseriti nella riga, ad esempio:

```
_branoCorrenteRow.Anno  =_bReader.ReadInt32();
_branoCorrenteRow.BitRate = _bReader.ReadInt32();
_branoCorrenteRow.SampleRate = _bReader.ReadInt32();
_branoCorrenteRow.Volume = _bReader.ReadInt32();
```

mentre altri vengono semplicemente estratti e messi in variabili non utilizzate. Questo perché, in futuro si potrebbe voler utilizzare questi valori. Altri byte vengono estratti tramite il metodo *ReadBytes* del *BinaryReader* e non vengono memorizzati: in tal caso i byte non contengono informazioni significative o conosciute. Questa strategia viene mantenuta anche nelle altre funzioni di estrazione. Dalla lettura fatta si conosce il numero di stringhe che contengono informazioni per un dato brano: utilizzeremo questa informazione al momento di estrarre le stringhe per decidere quando sia giunto il momento di aggiungere la riga alla tabella e passare al prossimo brano.

```
/// <summary>
/// estra i dati da una sezione stringa
/// </summary>
private void EstraiSezioneMhod()
{ int lunghezzaSezione=_bReader.ReadInt32();
   int lunghezzaSezione100 = _bReader.ReadInt32();
 ...}
```

La sezione stringa (*mhod*) contiene una stringa Unicode che assume differente significato a secon-

I FILE NEL CD

ITUNESEXTRACTOR.CS : la classe che effettua l'estrazione dei dati

IPODDS.XSD, IPODDS.CS: i file del DataSet tipizzato

tali file devono essere aggiunti alla eventuale applicazione. Si noti che il namespace delle classi è *TestApplp*. Per importare il dataset potrebbe essere necessario creare un nuovo dataset e copiare in questo le tabelle del dataset *ipodDS.xsd*.

da del suo tipo. Nel codice si vede come questo sia stato sfruttato per decidere a quale campo dell'oggetto *_branoCorrenteRow* o *_playlistCorrenteRow* assegnare la stringa estratta. Per quanto riguarda invece le playlist, utilizziamo le funzioni seguenti:

```
/// <summary>
/// Legge la sezione Mhyp per ogni Playlist
/// </summary>
private void EstraiSezioneMhyp()
{   if (!_inLetturaPlaylist)
  { _playlistCorrenteRow =
                              _DS.PlayList.NewPlayListRow();
  }
  int lunghezzaSezione = _bReader.ReadInt32();
  int lunghezzatotale = _bReader.ReadInt32();
  int indice = _bReader.ReadInt32();  //non univoco??
  //se di tipo 1 è la playlist principale
  _playlistCorrenteRow.quantibrani =
                                     _bReader.ReadInt32();
  _playlistCorrenteRow.tipo = _bReader.ReadInt32();
  _bReader.ReadBytes(12);
  _playlistIndex++;
  _playlistCorrenteRow.index = _playlistIndex;
  _bReader.ReadBytes(lunghezzaSezione - 36); }
  /// <summary>
 /// un brano nella playlist
 /// </summary>
 private void EstraiSezioneMhip()
 {  _playlistBranoCorrenteRow =
                           _DS.RelazionePlayListBrano.
                        NewRelazionePlayListBranoRow();
   _playlistBranoCorrenteRow.playlistindex =
                              _playlistCorrenteRow.index;
   int lunghezzaSezione = _bReader.ReadInt32();
   int altro = _bReader.ReadInt32();
   _bReader.ReadBytes(8);
   int correlazione = _bReader.ReadInt32();
   _playlistBranoCorrenteRow.branoindex =
                                     _bReader.ReadInt32();
   int sconosciuto = _bReader.ReadInt32();
   _DS.RelazionePlayListBrano.Rows.Add(
                             _playlistBranoCorrenteRow);
   _bReader.ReadBytes(lunghezzaSezione - 28);
}
```

Queste funzioni, rispettivamente, estraggono i dati di una playlist e di un brano che le appartiene. Una volta estratti i dati e popolato il *DataSet* potremo utilizzare i dati contenuti nel campo *Path* delle righe della tabella *Brano* per estrarre dal nostro iPod i brani, semplicemente effettuando una normale copia fra file, per esempio data una riga *RigaBranoRow* che contiene i dati di un brano, e avendo in *iPodDrive* il nome dell'unità disco assegnata all'iPod, il seguente codice copia il brano sul disco C:

```
string PathOriginale= iPodDrive+":
\\"+RigaBranoRow.Path;
string Estensione=RigaBranoRow.Path.Substring(
                        RigaBranoRow.Path.Length-3,3);
File.Copy(PathOriginale, @"c:\"+RigaBranoRow.Titolo+
                                       "." +Estensione);
```

CONCLUSIONI

È interessante notare che l'aver utilizzato un DataSet per i nostri dati ci permette di scrivere un'interfaccia sofisticata per l'applicazione che utilizza la nostra classe, mostrando i brani sotto le viste cui siamo abituati in iTunes o nell'iPod stesso. Per esempio, se nella nostra applicazione volessimo popolare una combobox con le playlist trovate potremmo scrivere un codice del genere:

```
iTunesDBExtractor itb= new iTunesDBExtractor();
itb.PercorsoDB=@" O:\iPod_Control\iTunes\itunesDb";
                                  //o il nostro path!!!
ipodDS _ds= _ds=itb.ExtractDS();
comboBox1.DataSource=_ds.PlayList.Select("","tipo
desc");
comboBox1.DisplayMember="nome";
```

mentre per popolare una datagrid a partire dalla playlist selezionata, potemmo scrivere nel gestore d'evento appropriato:

```
ipodDS.PlayListRow P=_ds.PlayList.Select(
     "nome='"+comboBox1.Text.Replace("'","''")+"'")[0]
                                    as ipodDS.PlayListRow;
ipodDS.RelazionePlayListBranoRow[] Relazioni=
                      P.GetRelazionePlayListBranoRows();
ListViewItem LVI;
ListViewItem[] Items= new ListViewItem[
                             Relazioni.GetLength(0)];
int idx=0;
foreach (ipodDS.RelazionePlayListBranoRow Riga in
                                            Relazioni)
{ LVI=new ListViewItem();
  LVI.SubItems.Add(Riga.BranoRow.Titolo);
  LVI.SubItems.Add(Riga.BranoRow.Path);
  LVI.SubItems.Add(Riga.BranoRow.IsArtistaNull()
                            ?"":Riga.BranoRow.Artista);
  LVI.SubItems.Add(Riga.BranoRow.IsAlbumNull()?
                              "":Riga.BranoRow.Album);
  Items[idx++]=LVI; }
  listView1.Items.Clear();
  listView1.Items.AddRange(Items);
```

A questo punto è facile utilizzare le tecniche standard di filtro dei dati per far sì l'interfaccia dell'applicazione sia completa e accattivante quanto vogliamo. Buon divertimento a tutti gli iPod-maniaci!

Marco Poponi

www.apple.it/ipod
per la struttura del file iTuneDB, ci sono numerose fonti, fra cui quelle del progetto
http://sourceforge.net/projects/ipod-on-linux/

Spiegazione header
http://sourceforge.net/docman/display_doc.php?docid=11191&group_id=52976

Introduzione agli header
http://sourceforge.net/docman/display_doc.php?docid=11212&group_id=52976

Marco Poponi è laureato in Ingegneria Elettronica. Insegna Lab di Programmazione alla facoltà di Ingegneria dell'Università degli Studi di Perugia. È tra i fondatori di InnovActive Engineering (http://dotnet.innovactive.it), azienda di sviluppo software e consulenza specializzata nella tecnologia .NET. Si occupa di progettazione, sviluppo e formazione.

Realizzazione degli “oggetti” di Sierpinski in 2D e 3D

Costruzioni geometriche iterative

Le figure di Sierpinski rappresentano un’interessante forma d’arte e sono alla base delle teoria per la costruzione dei frattali. Un esempio di geometria che trova la sua massima espressione con i PC

“O*rdiniamo il caos”*. Dietro tale ossimoro non si cela un banale controsenso, ma il tentativo di chiarire una materia ricca di teoria e applicazioni. Ci prefiggiamo di studiare e analizzare un ben circoscritto ambito del caos.
Abbiamo più volte affrontato l’argomento, in questo spazio della rivista, con particolare attenzione all’applicazione più allettante: i frattali. Nel presente appuntamento non parleremo di frattali, anche se in fondo ne svilupperemo uno, ma concentreremo le nostre attenzioni verso aspetti geometrici. Costruiremo graficamente nuovi oggetti con opportuni algoritmi. In particolare, ci dedicheremo allo studio delle forme di Sierpinski, con l’obiettivo finale di costruirne di interessanti nello spazio 3D. Un esempio suggestivo è riportato in Fig. 1.

***Fig. 1:** Una suggestiva rappresentazione del cubo di Sierpinski*

Ribadisco ancora una volta come la soluzione di problemi di questo tipo non sia soltanto un esercizio per sollecitare la personale corda estetica, ossia una pura attività artistica, ma si tratti spesso dello zoccolo teorico per lo sviluppo di una più ampia casistica di problemi. Quindi, se è vero che gli output che ci apprestiamo a produrre, le forme di Sierpinski, sono vere e proprie forme d’arte, peraltro presenti in regolari mostre sia tradizionali che su web; è altrettanto una realtà che sono stati un utile strumento per lo studio di fondamentali concetti della matematica moderna, come l’autosimilarità e la dimensione frazionaria. Tenterò, nei prossimi paragrafi, di fornire tutte le nozioni necessarie per raggiungere lo scopo prefissato, ovvero, la produzione di forme 3D. Durante il percorso ne approfitteremo per esplorare importanti nozioni teoriche .

IL PRIMO TENTATIVO

Una significativa base teorica all’argomento è fornita dal contributo del famoso matematico Cantor. Egli, con un semplice quanto esplicativo procedimento, costruì un insieme di infiniti elementi. Per generare l’*insieme di Cantor* si deve considerare un segmento di lunghezza *1*, i cui estremi sono i numeri *0* e *1* (compresi). Tale segmento viene diviso in tre parti; la parte centrale rimossa. Ai due segmenti rimanenti si applica la stessa tecnica, cosicché, iterando si ottiene man mano un maggior numero di segmenti di lunghezza sempre minore, come mostrato in Fig. 2. Alla generica iterazione *i*, il numero di segmenti è *2i,* mentre la lunghezza di ognuno di essi è pari a *1/3i*. L’*insieme dei terzi centrali di Cantor* o maggiormente conosciuto come l’*insieme di Cantor* è quindi prodotto dai segmenti che si hanno dopo infinite iterazioni. La continua foratura dei segmenti contrariamente a ciò che saremmo indotti a pensare, non produce un insieme vuoto, bensì infiniti punti. La dimostrazione, per la sua logica discreta, ha affinità con la programmazione. Poiché ad ogni passo, per un generico segmento ne vengono prodotti due nuovi, allora ogni estremo può essere individuato da un percorso, ossia una sequenza di riferimenti a destra *(D)* o a sinistra *(S)*. Semplificando, una sequenza di *S* e *D*. Ad esempio, *1* viene individuato da una successione di tutte *D*, poiché indica sempre il seg-

***Fig. 2:** Insieme di Cantor*

mento di destra. Il numero *1/3* (si veda la Fig. 2) è dato dalla sequenza *SDDDD…* è cioè sempre l'estremo destro a partire dal primo segmento sinistro. La dimostrazione di Cantor si basa sull'esame della sequenza oscillatoria *SDSDSDSD…* che non è un estremo, poiché non termina in una successione costante di tutte *D* o di tutte *S* (proprietà di un estremo di intervallo), mentre è un punto dell'insieme proprio, perché corrisponde ad una successione di *D* e *S*. Il punto vale *1/4* poiché è una serie matematica il cui risultato è noto. È adesso chiaro come si genera l'*insieme di Cantor*. Di seguito è proposto un metodo formale che sarà utile nei prossimi paragrafi.
Ogni elemento dell'insieme (nel caso specifico un generico segmento) detto assioma, viene sostituito da un generatore. Iterando il procedimento di sostituzione di assiomi con generatori si perviene all'insieme descritto. L'assioma è il segmento pieno, mentre il generatore (il cosiddetto segmento forato) è costituito da due segmenti e uno spazio centrale; tutti elementi di lunghezza pari ad un terzo dell'assioma (Fig. 3). Ovviamente, la coppia assioma-generatore si intende in scala e non di lunghezza fissa, per cui alla prima iterazione assioma avrà lunghezza *1*, alla seconda *1/3*, alla quarta *1/9* e così via.

SOFTWARE
Nel CD, oltre al programma presente sull'articolo, sono raccolti dei programmi sviluppati per altri articoli correlati.

Assioma
Generatore

Fig. 3: Assioma e generatore per l'insieme di Cantor

VERSO IL CAOS

Adesso proviamo a costruire una figura geometrica iterativa. Un interessante metodo per crearla in uno spazio a due dimensioni fa uso della teoria del caos. Si considerano tre punti *P1, P2* e *P3* di un triangolo (se equilatero il procedimento è di più facile intuizione), come mostrato in Fig. 4. Cominciando da un punto *0* scelto a caso nel piano, si considera uno dei tre vertici, ancora una volta casualmente. Supponiamo che il risultato di tale scelta aleatoria abbia prodotto il punto *P1*. Si traccia allora una linea immaginaria tra il vertice *P1* e il punto *(0)*, e si riporta il punto mediano; per intenderci diamo ad esso il nome *1*. Il processo viene iterato a partire dal punto *1*.
Per chiarezza, descriviamo un altro passaggio. Viene scelto casualmente uno dei tre vertici, questa volta *P2,* si traccia il segmento di congiunzione tra i due punti e si riporta il punto mediano *2;* e così via.
La funzione *random* per l'esempio proposto in Fig. 4 ha generato la sequenza di vertici: *1,3,3,1,2*. Nel grafico realizzato solo per comodità sono stati disegnati i segmenti di congiunzione dei punti; al fine della costruzione grafica tali segmenti sono solo immaginari. Soltanto i punti verranno realmente tracciati. Perché caos? Ci si potrebbe chiedere. Semplicemente perché si tratta di una costruzione geometrica che fa riferimento ad elementi generati casualmente, quindi in modo caotico. Ma avremo modo di approfondire altri aspetti della questione. Soffermiamoci un attimo sull'output prodotto da tale procedimento. È facile intuire come i punti descritti sono tutti interni al triangolo. È un po' meno evidente che si creeranno delle aree vuote, alcune anche consistenti in cui non saranno presenti punti. La produzione grafica in varie fasi del processo di costruzione è riportata in Fig. 5. Abbiamo appena sviluppato una distribuzione di punti conosciuta come triangolo di Sierpinski.

PER SAPERNE DI PIÙ

RIFERIMENTI SU IOPROGRAMMO
Nella sezione *Soluzioni* di ioProgrammo, sono un utile riferimento gli articoli sui frattali ai numeri: 50, 51 e 52 quelli sulla grafica reperibili ai numeri: 36 e 37.

Fig. 4: Metodo per la costruzione di figure geometriche iterative mediante teoria del caos

SIERPINSKI IN 2D

Un elemento che appare ricorrente, nonché filo conduttore, è il concetto di iterazione. Con la continua ripetizione dello stesso procedimento si raggiunge l'obiettivo, ovvero la costruzione di particolari figure geometriche. Come vedremo, una conseguenza sarà la produzione di forme autosimilari.
Ma, come mostrato in Fig. 5, si può notare come il processo si completi con numeri di iterazioni sempre maggiori. Teoricamente bisogna far tendere il numero di iterazioni all'infinito. Il programma pascal proposto di seguito costruisce il triangolo di Sierpinski con centomila iterazioni. Come accortezza, ereditata dalle tecniche per lo sviluppo di frattali, vengono scartate le prime mille iterazioni; nel linguaggio proprio dell'ambito dei frattali si parla di *orbite* (e non iterazioni).

```
Program Triangolo_di_Sierpinski;
Uses graph;
var D,G,r : Integer;
    x,y : real;
      i : longint;
Begin
   D:=Detect;
   Initgraph(D,G,'c:\tp\bgi');
   randomize;
   x:=40; y:=50;
   For i:=1 to 100000 do
   Begin
      r:=random(3)+1;
      case r of
      1 :  Begin
           x:= (400+x)/2;
           y:= (400+y)/2;
         End;
      2 :  Begin
           x:= (200+x)/2;
           y:= (53+y)/2;
         End;
      3 :  Begin
           x:= x/2;
           y:= (400+y)/2;
         End;
      End;
```

```
        if i>1000 then Putpixel(round(x),round(y),YELLOW);
    End;
    readln;
    Closegraph;
End.
```

Fig. 6: ***Generazione geometrica del triangolo di Sierpinski fasi- triangolo di Sierpinski***

Il punto iniziale è stato fissato a (40,50), si poteva anche generarlo casualmente. Mentre i vertici sono stati scelti per avere un buon effetto su un tavola grafica di 640 x 480 punti; essi hanno coordinate rispettivamente pari a: (200,53), (0,400) e (400,400). La figura appena ottenuta con il programma pascal può essere generata geometricamente. Si tratta di applicare il processo iterativo di sostituzione assioma-generatore, già esaminato per l'*insieme di Cantor.* In Fig. 6 vengono riportati i due elementi: assioma e generatore, nonché il risultato ottenuto dopo la quarta iterazione.

Un altro metodo per la generazione del triangolo è stato sviluppato da Lindenmayer che ha attuato il sistema omonimo. Ad ogni iterazione, gli assiomi (qui linee, in tale sistema si focalizza l'attenzione sulle linee e non sui triangoli), vengono sostituiti da generatori che ad ogni passo vengono opportunamente scalati. Il sistema è riportato in Fig. 7. L'assioma è un solo lato del triangolo. Esaminando la figura si comprende chiaramente il processo di costruzione. Un lato di un triangolo viene sostituito dal generatore opportunamente scalato, i restanti due lati vengono riportati così come sono. Ovviamente, a tale procedimento vanno sottoposti tutti i triangoli presenti nella figura in esame.

Esistono molti sistemi di Lindermayer che possono generare gradevoli figure geometriche, come quella riportata in Fig. 8, che ricorda il triangolo di Sierpinski pur essendo diversa. La figura ricorda anche un merletto che è tanto più rifinito quanto sono

Fig. 7: ***Sistema di generazione di Lindenmayer***

maggiori le iterazioni. Per chi volesse approfondire lo specifico aspetto consiglio di visionare anche il fiocco di neve di Van Kock, riferitevi agli articoli correlati proposti nella rivista. Oltre ai due metodi visionati per la generazione del triangolo di Sierpinski; il primo che fa uso della teoria del caos e che è stato implementato da un programma pascal, e il secondo attraverso una costruzione puramente geometrica che sfrutta la produzione di un generatore applicato ad un assioma, ne esiste un terzo. Sicuramente si potranno individuare anche altri metodi, ma esaminiamo questa semplice idea. Si considera il triangolo si Tartaglia, anche conosciuto come triangolo di Pascal (nome ricorrente in questo appuntamento!). Per chi non lo ricordasse è usato per stabilire i coefficienti dei termini della potenza dei binomi. Esso assume la forma:

1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

Fig. 8: ***Sistema di Lindenmayer per figura a merletto***

NOTA

SIERPINSKI

Waclaw Sierpinski nacque il 12, marzo 1882 a Varsavia, quando la città era sotto l'occupazione della Russia. I Russi avevano imposto la loro lingua e la loro cultura a tutte le scuole secondarie della Polonia e preferivano che i polacchi restassero analfabeti, tanto che il numero di studenti era crollato. Nonostante le difficoltà, Sierpinski entrò nel dipartimento di matematica e fisica dell'Università di Varsavia nel 1899. Nel 1903 vinse anche una medaglia d'oro per un suo saggio sulla teoria dei numeri, ma, non volendo che fosse pubblicato in russo, attese fino al 1907 quando fu edito in inglese. Rischiò di non ottenere la laurea in scienze matematiche perché, volontariamente, non superò l'esame di russo; per i suoi meriti scientifici tuttavia l'insegnante di russo cambiò in "buono" il pessimo risultato del suo esame, ed egli ottenne la laurea. Si occupò di molteplici studi: dalla teoria degli insiemi, ai numeri irrazionali, all'astronomia, alla filosofia. Anche la 2° guerra mondiale lo segnò e fu costretto alla clandestinità. Egli si ingegnò a spedire in Italia le sue carte affinché venissero pubblicate, a questo periodo risalgono gli studi sui frattali. Dopo la rivolta del 1944 i nazisti incendiarono la sua casa, distruggendo la sua biblioteca e tutti i documenti personali. Capace di lavorare in qualsiasi condizione, raggiunse infine il meritato successo. Tanti sono i riconoscimenti attribuitigli. Morì il 21, ottobre, 1969 a Varsavia.

Per costruire il triangolo di Sierpinski sulla base di quello di Pascal bisogna analizzare i singoli elementi di questo secondo. Gli elementi dispari faranno parte del triangolo, mentre i pari verranno rimossi. Si ottiene così un triangolo numerico di Sierpinski che, con opportune associazioni *numero con porzione di spazio,* può dare origine anche a un triangolo in forma geometria dei tipi precedentemente prodotti. Chiudiamo la rassegna 2D con il quadrato di Sierpinski, conosciuto anche come *tappeto*, che come vedremo oltre al solito effetto grafico, ancora una volta piacevole, mostra altri motivi di interesse.

Per evitare di dilungarci, consideriamo la sola produzione geometrica che è riportata in Fig. 9. Notiamo una similitudine tra il quadrato di Sierpinski, realizzato in uno spazio 2D, e l'insieme di Cantor (Fig. 2). Nel procedimento di Cantor si rimuove alla prima iterazione il segmento centrale, nel tappeto di Sierpinski il quadrato centrale, e così l'analogia continua tra tutti i segmenti del primo procedimento e i quadrati del secondo. Come vedremo, anche nello spazio 3D si potrà rilevare tale similitudine.

ALCUNE COSE DA SAPERE

Fig. 9: ***Quadrato di Sierpinski***

Prima di passare al momento clou del nostro appuntamento, ovvero l'esame di forme 3D, è necessario soffermarsi su alcuni fondamentali concetti teorici. Cominciamo con l'autosimilarità, caratteristica comune a tutte le forme finora costruite. Si tratta di una proprietà tipica di molte immagini frattali. Si presenta quando una porzione di figura è simile ad un'altra porzione della stessa figura che la contiene o che è contenuta. Più precisamente, definiamo due figure come simili se hanno la stessa forma, ovvero, se sono uguali a meno di una traslazione e/o un ingrandimento (o riduzione). Tale risultato si ottiene attraverso una trasformazione lineare. Quando questi caratteri di "somiglianza" sono in un'unica figura, cioè una parte di essa è simile ad un'altra parte più grande (quindi è uguale a questa ultima a meno di un ingrandimento e di una traslazione), allora diremo che tale figura gode della proprietà di autosimilarità. Altro concetto da conoscere per chi ha intenzione di approfondire l'argomento è la dimensione di tali figure, che non è necessariamente un numero intero come siamo abituati a pensare. Un modo per ottenere tale valore da una figura geometrica fa uso del concetto di autosimilarità appena studiato. Un segmento, un quadrato o un cubo sono tutte figure autosimilari. Un segmento di lunghezza unitaria può essere diviso in n parti ciascuna di lunghezza *1/n*, facendo un'analoga scomposizione per il quadrato si ottengono *n2* quadratini di area *1/n2* e così *n3* cubetti di volume *1/n3*. La dimensione dei tre oggetti è data dall'esponente di *n*, ecco un metodo semplice e veloce! Sebbene per le figure trattate il procedimento sia banale, per i frattali e le figure che stiamo studiando non è così. Procediamo formalizzando tale tecnica: determiniamo il logaritmo del numero di elementi che servono a comporre l'oggetto nei tre casi esposti.

log(n elementi)= log(n1)=1log(n)
log(n elementi)= log(n2)=2log(n)
log(n elementi)= log(n3)=3log(n)

Poiché le figure estratte restituiscono quella iniziale se ingrandite di un fattore *n*, allora la dimensione è il rapporto tra i logaritmo del numero di elementi e il logaritmo di *n*. Cosicché, per le figure elementari *segmento, quadrato* e *cubo* le dimensioni *D* sono:

D=log(n elementi)/log(n)=log(n1)/log(n)=1log(n)/log(n)=1
D=log(n elementi)/log(n)=log(n2)/log(n)=2log(n)/log(n)=2
D=log(n elementi)/log(n)=log(n3)/log(n)=3log(n)/log(n)=3

Se applichiamo tale procedimento al triangolo e al quadrato di Sierpinski si avranno le seguenti dimensioni:

Dts = log(n elementi)/log(n)= log(3)/log(2)=1.585
Dqs = log(n elementi)/log(n) = log(8)/log(3)=1.893

Ricordo, infatti, che il triangolo viene diviso in tre triangolini e che il fattore di ingrandimento per ripristinare l'immagine dell'iterazione precedente è *2*. Mentre, il quadrato è diviso in otto quadratini con fattore di ingrandimento, per la ricostruzione dell'immagine, *3*.
A sorpresa, i risultati sono numeri razionali.

OBIETTIVO RAGGIUNTO

Passo conclusivo dell'intero percorso è la produzione di oggetti tridimensionali. In questo ambito, le abilità artistiche e la fantasia di chi si cimenta nella creazione di forme 3D vengono esaltate dando vita a sorprendenti realizzazioni. Spesso si possono ammirare tali produzioni in mostre d'arte moderna. Ma passiamo all'aspetto tecnico. Il procedimento per la costruzione segue le stesse regole del contesto bidimensionale. La differenza sostanziale nelle due circostanze riguarda l'elemento oggetto della costruzione; nel caso bidimensionale è un triangolo, mentre in 3D è una piramide. Attuiamo una costruzione geometria facendo uso di assioma e generatore. In Fig. 10 è riportato, oltre che il risultato finale, anche la coppia assioma generatore. Nella stessa figura, nella sezione *b*, si può osservare il me-

Fig. 10: ***Costruzione geometrica della piramide e del cubo***

todo applicato alla costruzione del cubo. Un'analisi dimensionale, in comparazione con il contesto bidimensionale, evidenzia che entrambe gli oggetti ottenuti hanno volume *zero* e un'infinita superficie. Il risultato si può anche dimostrare. Come era prevedibile le analogie nel passaggio dal 2D al 3D non si fermano qui. Ad esempio, si può notare come le quattro facce della piramide siano dei triangoli di Sierpinski, così come le sei facce del cubo corrispondano al tappeto. Nel caso della piramide ogni oggetto si scompone in altri cinque, mentre, il fattore di ingrandimento per rendere un oggetto di eguale grandezza con il suo generatore è due. Questi due numeri ci permettono di conoscere anche per la piramide la sua dimensione.

Dp = log(n elementi)/log(n)= log(5)/log(2)=2.322
Dc = log(n elementi)/log(n) = log(20)/log(3)=2.727

Con *Dc* indichiamo la dimensione del cubo. Lo sviluppo di un algoritmo che riporti graficamente un oggetto tridimensionale come la piramide, può essere attuato sulla base della teoria del caos, in analogia al programma sviluppato per il caso 2D. I vertici di cui tener conto sono quattro, cosicché la scelta multipla sarà su quattro possibilità; per ognuna di esse si dovrà calcolare il punto mediano tra il punto corrente e il vertice individuato casualmente. Al termine del calcolo non resta che disegnare il punto sullo schermo. Questa volta non è possibile farlo direttamente come nel caso 2D, con la sola chiamata alla funzione *putpixel* (usata in pascal, il linguaggio scelto per il nostro esempio), sarà necessaria una trasformazione da tre a due dimensioni. In altri termini la terna *x,y,z* sarà trasformata nella coppia *x1,y1*. Per farlo si possono adoperare le conosciute matrici di rotazione, con le quali stabilire anche i punti di vista dell'osservatore fissando due coefficienti. Gli oggetti geometrici iterativi costruiti fino a questo momento non superano dimensione 3. È comunque possibile andare oltre è produrre degli iperoggetti. Un metodo intuitivo per determinare la geometria 4D di una piramide (o meglio iperpiramide) fa riferimento allo stesso oggetto in uno spazio ad una dimensione minore. L'oggetto in dimensione minore, quindi 3D, sarà definito sulla base delle informazioni della dimensione ancora minore di 1, ossia 2D; e per finire si farà riferimento allo spazio 1D. Nello spazio bidimensionale un triangolo viene costruito prendendo un segmento, in 1D, come base e "tirando" il punto medio in modo da portarlo in 2D. Il verbo *tirare* non è a rigore corretto, ma ha il pregio di rendere chiara l'idea, come si potrà verificare dalla rappresentazione in Fig. 11. Analogamente una piramide può essere ottenuta "tirando" una figura piana come un rombo (Fig. 11 b). In definitiva, la costruzione di un oggetto a dimensione *N* può essere codificata come una serie di compiti elementari:

- Partire con un oggetto a dimensione *N-1* centrato nell'origine;
- Tirare il punto medio in modo da generare un oggetto di dimensione *N;*
- Costruire gli spigoli congiungendo il punto medio a ogni vertice dell'oggetto di dimensione *N-1;*
- Costruire le facce usando il punto medio e ogni spigolo del oggetto di dimensione *N-1*.

Usando queste regole, si può costruire un iperpiramide nello spazio 4D a partire da una piramide in 3D. Alla prima iterazione, la base contiene otto iperpiramidi di cui una all'apice. Ad ogni iterazione il numero di iperpiramidi cresce di un fattore 9, cosicché, la dimensione è *log(9)/ log(2)* pari a 3.17. Il problema è quello di visualizzare l'iperoggetto in uno spazio graficamente trattabile come 2D o 3D. Le curve di livello sono un modo per visualizzare un oggetto 3D in un piano; in 2D si riporta il livello dell'oggetto, come se questo ultimo fosse intersecato da un piano ad una determinata altezza. Così si disegna l'intersezione tra il piano e l'oggetto. Una pila di *N* linee di livello da una rappresentazione dell'oggetto in 3D. Analogamente, si può procedere nel passaggio da 3D a 4D. In tal caso anziché parlare di linee di livello, si dovrà trasporre il termine in solidi di livello.

BIBLIOGRAFIA

- **STUDIES IN GEOMETRY *Blumenthol Leonard, Menger Karl* (W.H.Freeman & Co) 1970**
- **GRUNDLAGEN EINER ALLGEMEINEN MANNICHFALTIGKEITSLEHRE *G. Cantor* (Mathematische Annalen 21) p545-591, 1882**
- **THE FRACTAL GEOMETRY OF NATURE *Mandelbrot Benoit* (W.H.Freeman & Co) NY, 1982**
- **GLI OGGETTI FRATTALI *Benoit Mandelbrot* (Einaudi) 1987.**
- **THE SCIENCE OF FRACTAL IMAGES *H. Peitgen, D. Saupe* Springer-Verlag, NY, 1988**
- **LECTURE NOTES IN BIOMATHEMATICS *Przemyslaw Prusinkiewcz, James Hanan* (Springer-Verlag) NY, 1980**
- **FRACTAL AND DISORDERD SYSTEMS Armin *Bunde/Shlomo Halvin (ed.)* (Springer) 1991.**

Fig. 11:* *Costruzione di oggetti a partire da figure in spazi di una dimensione minore

CONCLUSIONI

Come abbiamo potuto constatare l'argomento è ricco di spunti teorici e prevede diverse possibili applicazioni. Inoltre, si possono costruire altre figure geometriche, basta avere un po' di fantasia e dimestichezza con il metodo. Infine, un aspetto che merita approfondimento è lo studio degli oggetti negli iperspazi; è questo uno spunto per un futuro articolo.

Fabio Grimaldi

Giocare con numeri interi e lettere

Il gioco del 15

Continua la rassegna di enigmi e giochi che si fondano sulle comuni proprietà dei numeri. Il gioco del 15 è semplice quanto sorprendente e appartiene ad una più ampia famiglia, il cui capostipite è il quadrato magico

NUMERI INTERI

L'insieme dei numeri interi, indicato in matematica con *N*, rappresenta gli infiniti elementi compresi nell'intervallo aperto *] 0 , infinito [*, ossia i numeri (senza la virgola): *0,1,2... infinito.* In programmazione con il termine *intero*, associato a tipi di variabile, si intende qualcosa di leggermente diverso, cambia ad esempio il range di definizione, che per limiti legati alla macchina non può essere infinito.

Un foglio di carta o meglio, un blocco note e una penna sono gli unici due strumenti che serviranno per cimentarsi nello svelare gli enigmi di questa puntata. Il protagonista dello spettacolo è sempre il numero, nella sua semplicità e forza. Basterà sfruttare le sue caratteristiche di base per accedere alla chiave di tutti i giochi in scena. Il nuovo enigma parlerà di zuppe. Fatti i dovuti approfondimenti, il sipario si chiuderà su un grande rompicapo che conosciamo bene: il quadrato magico. Quindi, se avete in mano il sacchetto dei pop-corn, possiamo cominciare.

IL GIOCO DEL DIVISORE

Ricordo che il gioco vede due contendenti che, a turno devono scrivere su un foglio un numero, secondo prefissate regole. Perde chi non ha alcun numero da poter scrivere; il vincitore sarà il suo avversario. Si parte con un numero generato casualmente o, alternativamente, proposto da uno dei due partecipanti, in tal caso, per più partite, sarà un compito assegnato a turno. Il giocatore a cui spetta la mossa, deve individuare per il numero corrente (al primo turno l'unico scritto sul foglio) un suo divisore, e scrivere la differenza tra i due. Il suo avversario dovrà fare la stessa cosa, con il nuovo numero appena riportato. Si continua così fin quando uno dei due giocatori perde, allorquando si ritrova un numero primo a cui non può sottrarre alcun divisore. Per un fissato numero non sono ritenuti divisori validi se stesso e uno. Spero che nel corso di questo mese gli appassionati del genere abbiano approntato una strategia vincente. Essendo il gioco molto semplice, individuare una strategia adeguata non risulta molto difficoltoso, si tratta di tenere presente alcune proprietà dei numeri. L'osservazione principale è che un numero dispari può avere soltanto un divisore dispari. Da ciò si deduce che un numero dispari è sempre seguito da uno pari (la sottrazione tra due dispari genera sempre un pari). Poiché il gioco non può terminare con un numero pari, allora un giocatore che se lo ritrova non può perdere, almeno in quel turno. Quindi, la strategia impone di lasciare al nostro avversario, se possibile, un numero primo (ma questa sarebbe una situazione alquanto favorevole), oppure, un numero dispari. Questa seconda considerazione ci porta ad avere sempre un numero pari da trattare al nostro turno, che garantisce l'attuazione della strategia vittoriosa appena descritta. L'analisi approfondita avrà sicuramente deluso qualcuno per la cruda realtà emersa, la quale evidenzia che, se entrambi i giocatori conoscono a fondo la strategia, il vincitore sarà il primo giocatore, se il numero iniziale è pari, il secondo se dispari. Un risultato che dipende fortemente dal caso. Una variante del gioco che non modifica la sostanza, permette ai giocatori di considerare uno (1) come divisore. In questa versione si deve avere la sola accortezza di lasciare sempre un numero dispari all'avversario.

IL GIOCO DEL 15

Ancora un semplice e significativo gioco che si basa su numeri interi. I contendenti al solito sono due è gli strumenti essenziali sono sempre la carta e la penna. Alla strumentazione spartana di un foglio puramente bianco del gioco del divisore, per la nuova attrazione, si aggiunge una griglia, ossia una sorta di vettore di nove caselle numerate da uno a nove. A turno, i due sfidanti devono scegliere una delle caselle, evitando di occupare quelle già segnate. Come nel gioco del tris, i due giocatori possono scegliere uno tra due simboli per marcare le caselle, ad esempio, un cerchio e una croce. Il numero segnato da ognuno dei partecipanti si va a sommare al suo personale punteggio che inizialmente sarà ovviamente zero. Lo scopo del gioco è quello di totalizzare 15. Simuliamo una partita per comprendere meglio. Supponiamo che due amici: *Tizio* e *Caio*, decidano di sfidarsi al gioco del 15. Nelle varie partite che disputeranno dovranno cominciare alternativamente. Per la prima, supponiamo che tocchi a *Tizio*. Egli segna con la *X* il primo numero, per esempio 6. *Caio* risponde apponendo un cerchio sulla sua scelta, diciamo 9, per evitare che il suo avversario vinca al turno successivo. Dopo il primo turno il punteggi sono 6 per *Tizio* e 9 per *Caio*. *Tizio* al secondo turno sceglie 5. Ricordo che 6 e 9 non potevano essere tra le scelte essendo già marcati. *Caio* supponiamo marchi 2. Punteggio: 11 *Tizio* e 11 *Caio*. A questo punto *Tizio* vince segnando 4, infat-

ti, realizza 15. Un match può finire in parità, anzi se i due giocatori hanno ben capito le strategie vincenti, è una eventualità che capita spesso. Propongo come sfida di questo mese la pianificazione e realizzazione della giusta strategia per vincere. Come può entrare il computer nella questione? Sviluppando un programma che sia in grado di giocare contro l'uomo; oppure, più facilmente facendo un programma che consenta a due utenti di giocare. È questo ciò che realizza il semplice programma Pascal riportato si seguito. Data la relativa dimensione ridotta il listato è riportato integralmente.

```
Program verifica;
USES crt;
Const Dim=9;
Var ctizio, ccaio, turno, x: integer;
    campo,tizio,caio :string[DIM];
Begin
   clrscr;
   (* input *)
   write('Nome giocatore 1 : '); readln(tizio);
   write('Nome giocatore 2 : '); readln(caio);
   ctizio:=0; ccaio:=0; turno:=0;
   campo:='LLLLLLLLL'; (* Inizialmente le caselle
                                    sono tutte libere*)
   while (ctizio<>15) and (ccaio<>15) and (turno<9) do
   Begin
       turno:=turno+1;
       if (turno mod 2) = 0 then (*se pari il turno è
                              di tizio altrimenti di caio*)
       Begin
           clrscr;
           writeln('Ciao  ', tizio,' il tuo punteggio è
                                   ',ctizio,', tocca a te');
           writeln(campo);
           repeat
             writeln('Scegli il numero (tra 1 e 9)');
             readln(x);
           until campo[x]='L'; (* Bisogna scegliere
                                    solo caselle Libere*)
           campo[x]:='T';
           ctizio:=ctizio+x;
       End
       else
        Begin
           clrscr;
           writeln('Ciao  ', caio,' il tuo punteggio è ',ccaio,',
                                              tocca a te');
           writeln(campo);
           repeat
             writeln('Scegli il numero (tra 1 e 9)');
             readln(x);
           until campo[x]='L'; (* Bisogna scegliere solo
                                          caselle Libere*)
           campo[x]:='C';
           ccaio:=ccaio+x
       End;
   End;
   (*Analisi finale degli output*)
   if turno=9 then writeln('Non ci sono vincitori - pareggio')
   else if ctizio=15 then writeln('Il vincitore _ ', tizio)
        else writeln('il vincitore _ ',caio)
end.
```

La competizione si svolge sulla stringa *campo*, che inizialmente ha tutte le caselle libere (carattere *L*); i due giocatori, identificati dalle due stringhe *tizio* e *caio*, devono indicare il numero da segnare. Questo processo è codificato all'interno di un ciclo che termina con la vittoria di uno dei due o in caso di pareggio. I due contatori *ctizio* e *ccaio* mantengono il punteggio. Ogni qual volta i giocatori scelgono una casella viene segnata sul vettore campo (con la lettera *T*, per *Tizio*, e *C* per *Caio*). Infine, *turno* indica il corrente turno di gioco, esso serve sia per uscire dal ciclo in caso di parità e sia per dare alternativamente avvicendamento ai due contendenti. Per ragioni di spazio i controlli sono ridotti al minimo, ad ogni modo il programma è perfettamente funzionante.

LA ZUPPA È PRONTA

Una variante stuzzicante (e come vedremo appetitosa) del gioco appena visto è la *"zuppa di pesce"* o meglio *"soup fish"* con il suo nome anglosassone originale. Il gioco è simile al precedente, prevede lo stesso numero di partecipanti (due che agiscono alternativamente) è un campo analogo. La tabella riportata di seguito indica il vettore di gioco, che questa volta contiene parole. L'attenzione si è quindi spostata dal numero alla lettera.

VOTE	KNIT	ARMY	CHAT	FISH	SOUP	HORN	SWAN	GIRL

I due contendenti alternativamente devono scegliere una casella e marcarla, vince chi per primo totalizza il punteggio 3, ossia chi abbia nel suo set di parole tre con la stessa lettera. Esempio: una terna vincente è proprio una rivisitazione del titolo del gioco: *"soup swan fish"* per la presenza delle tre *S*; chissà tra l'altro se esiste una zuppa di pesce cigno. Ma è vincente anche *"army horn girl"* per le 3 *R*. Insomma, pur essendoci delle similitudini con il precedente gioco, la zuppa di pesce (il nome penso che sia stato dato per le assurde frasi che si ottengono, che ricordano una zuppa) richiede un maggiore acume tattico. Ma di questo e altro parleremo alla prossima puntata.

CONCLUSIONI

Per concludere l'appuntamento di oggi introduco un argomento che i fedelissimi di ioProgrammo conoscono: i quadrati magici; insieme ai due giochi precedentemente trattati saranno motivo di interesse per il prossimo appuntamento. Si tratta di matrici quadrate contenenti numeri interi appartenenti alla serie *1,2,3 .. n,* che hanno una particolare proprietà, la somma dei numeri, in orizzontale per ogni riga, in verticale per ogni colonna e nelle due diagonali dà sempre lo stesso numero.
Ovviamente tocca a noi incasellare i numeri.

Fabio Grimaldi

NOTA

GIOCO DEL TRIS

È un semplice gioco che molti conosceranno. Su una matrice quadrata, tre righe e tre colonne, due giocatori a turno devono segnare una delle caselle con il proprio simbolo, croce oppure cerchio. Vince chi riesce a fare tris, ovvero riporta una sequenza dei tre simboli: in orizzontale, o in verticale o anche in diagonale. Oltre oceano il gioco è conosciuto con il nome TicTacToe.

BIBLIOTECA

Il gioco del quindici è riportato da *M. Gardner* in alcuni suoi libri, mentre la zuppa si può trovare in

• YOUR MOVE
D.L. Silverman
(McGraw-Hill)
1971

INBox
L'esperto risponde...

Passaggio VB6 a VB Net

Ho di recente frequentato un corso di programmazione VB6 + SQL Server 2000 e, parlando con i miei docenti che sono dei programmatori che sviluppano software e tengono lezioni in merito, è venuto fuori il discorso del passaggio da VB6 al nuovo VB NET. Noi che frequentavamo abbiamo posto il problema se la programmazione in VB6 non fosse ormai obsoleta... loro, tra cui un docente che tiene seminari anche per Micro... (evitando pubblicità :-)) ci ha detto di stare ancora tranquilli che il passaggio dal vecchio al nuovo richiederà ancora un po' di tempo, in quanto le case di software che hanno sviluppato fino ad ora in VB6 non faranno subito il grande passo per motivi economici, infatti se ci si pensa su convertire tutto il programmato al nuovo .NET porterà via tempo e quindi per loro denaro.
Sarei molto interessato a sapere il vostro parere in merito a quanto da me esposto e dal vostro parere e se la vivete sul lavoro dalla vostra esperienza professionale.

Risponde Roberto Allegra

È vero: ci vorrà ancora tempo prima che VB6 venga abbandonato del tutto, sia per i comodi degli sviluppatori, sia per quelli delle aziende.
Fatto sta che è comunque un linguaggio in agonia, a cui è stato anche tolto ogni supporto ufficiale. E' un po' la situazione in cui si trova MFC, a parte il fatto che quest'ultimo è ancora integrato nei nuovi Visual Studio.
VB6 può essere sempre utile per imparare qualche principio di programmazione senza complicarsi troppo la vita, o per spiegare componenti e tecnologie in maniera semplice e accessibile a quante più persone possibile (questo è il motivo per cui scelgo, quando possibile, VB6 come piattaforma per i progetti trattati nei miei articoli).
Ma alla fine (uno, due anni), soprattutto all'uscita del famigerato Longhorn, che pare proprio non permetterà più il supporto a VB6, l'unica alternativa di sviluppo sarà .NET, in uno qualsiasi dei linguaggi compliant (VB,C#,C++).
E i programmatori devono pur prepararsi per tempo, non credi? Ci vuole tempo per assorbire qualla marea di classi fornite dalla *Class Library*, e per "gestire il cambiamento".
C'è anche da dire che, nel cambio VB6-VB.NET/C# non c'è praticamente nulla da perdere, e tutto da guadagnare: VB.NET è un linguaggio molto più robusto, orientato realmente agli oggetti, etc...
Il mio parere è: continua pure a sviluppare in VB6 se vuoi fare qualche progettino rapido, ma inizia a studiare e collaudare progetti in VB.NET e C#.
Su quale dei due linguaggi orientarsi, risponde ad altra questione...

Crystal da VB.NET

Sto lavorando ad una WinForm con il caricamento di un report. Se sul *SetDataSouce* carico la tabella, la mia stampa ha esito positivo, invece se caricato da un *DataSet* contente la tabella, visualizza il dataviewer vuoto.

```
Dim r As New CrystalDecisions.
    CrystalReports.Engine.ReportDocument
Dim dasTmp As New DataSet("Tmp")
dasTmp.Tables.Add(_datLis)
r = New rptListino
r.SetDataSource(dastmp)
CrystalReportViewer1.ReportSource = r
```

Qualcuno mi sa dire perchè? Ciao e grazie.

Paolo

Risponde Fabio Cozzolino

Classico problema: non hai letto il mio articolo sul numero 79 di ioProgrammo! In poche (pochissime) parole devi creare un dataset "tipizzato" ed utilizzarlo come sorgente dati del tuo report. Il dataset lo crei da Visual Studio .NET aggiungendo un nuovo file al progetto (nell'elenco troverai anche "dataset"). La spiegazione su come farlo è troppo lunga e non può essere espressa in poche righe, ti invito, perciò, ad effettuare una ricerca su google... Dal codice Vb.Net devi sostituire questa riga:

```
Dim dasTmp As New DataSet("Tmp")
```

con questa:

```
Dim dasTmp As New NomeMioDataSet()
```

C#: Qual è il mio indirizzo?

Vorrei sapere come si fa ad ottenere l'indirizzo IP della macchina?

Andrea Bello

Risponde Salvatore Meschini

Con il codice seguente, nella variabile *Indirizzi* avrai un vettore di indirizzi IP:

```
PHostEntry ComputerLocale =
  Dns.GetHostByName(Dns.GetHostName());
IPAddress[] Indirizzi =
                    ComputerLocale.AddressList;
```

Copiare tabella Access

Buongiorno ho un problema: devo creare una copia di una tabella contenuta in un DB Access, rinominarla e salvarla all'interno del DB stesso. Come posso fare... non sono molto esperta, quindi vi prego di essere molto clementi nelle spiegazioni. Ciao e grazie!

Paoletta

Rispond amdbook

La sintassi SQL per la copia della tabella dovrebbe essere la seguente:

```
SELECT Tabella.* INTO TabellaCopia FROM
                                   Tabella
```

...quindi se vuoi incapsulare il tutto all'interno di una routine di un *CommandButton*, il codice dovrebbere essere del tipo:

```
dim Conn as new Adodb.Connection
Conn.ConnectionString="Provider=Microsoft.
```

```
Jet.OleDb.4.0;Data Source=
PercorsoFileDatabase.mdb;"
Conn.Open
dim SQl as string
Sql="SELECT Tabella.* INTO TabellaCopia
FROM Tabella"
Conn.execute sql
Conn.Close
```

VB.NET: intercettare alt+e

Salve a tutti: vorrei che nella mia applicazione quando da tastiera si preme la combinazione *alt+e* il form si chiuda. Come posso fare?

device78

Risponde snogar

Tra le proprietà della Form c'è *KeyPreview:* mettilo a *True* e poi inserisci questo codice:

```
Private Sub Form1_KeyDown(ByVal sender
As Object, ByVal e As
System.Windows.Forms.KeyEventArgs)
Handles MyBase.KeyDown
If e.Alt And (e.KeyCode = Keys.E) Then
Me.Close()
End If
End Sub
```

MessageBox in Java con JBuilder

Ciao a tutti, qualcuno può indicarmi come si può avere l'equivalente di una messagebox in java con JBuilder? Inoltre, prima di posizionare controlli su una JDialog devo usare un *JPanel* ed è ok. Solo che questo pannello si posiziona in determinati punti della *JDialog* e quindi anche i controlli sono limitati nel posizionamento. Non c'e' la possibilità di avere a disposizione TUTTO il Dialog e non solo "North, Sud,..." ?

Luca

Risponde Krystal

Dai un'occhiata alla classe *JOptionPane*: nella stessa documentazione sono riportati alcuni esempi.
L'unica cosa che devi fare è impostare un LayoutManager diverso da quello di default (*BorderLayout*) o, nel tuo caso, settarlo a *null*... ma ti spiego meglio:
Nel dover definire la visualizzazione dei componenti grafici era necessario delegare a "qualcuno" il compito di gestire, secondo dei precisi criteri, la "disposizione" dei componenti. Tale compito è stato assegnato appunto ai *LayoutManager*, e in Java ce ne sono di diversi tipi:

• **BorderLayout**: consente la disposizione dei componenti in sei differenti regioni: *north, south, east, west*, e *center*

• **GridLayout:** consente la disposizione dei componenti in una griglia:

Ce ne sono tanti altri, puoi fare riferimento alla documentazione ufficiale di Java. Ogni componente è, a sua volta, un "container" destinato a contenere altri componenti, e possiede per questo un proprio *LayoutManager* che serve a "disporre" in un certo modo i figli, che a loro volta possiedono dei *LayoutManager* per disporre secondo altri criteri i componenti "nipoti", e così via...
Per cambiare *LayoutManager* ad un componente è necessario fargli invocare il metodo *setLayout* che riceve come argomento proprio un *LayoutManager*...
Nel tuo caso, quindi, o posizioni il tuo *JPanel* al "center" del *BorderLayout* che già ti ritrovi, o provi qualche nuova soluzione adottando diversi *LayoutManager* o semplicemente non ne imposti nessuno!
Per quest'ultima cosa basta richiamare il metodo *setLayout* passandogli come argomento *null:*

```
setLayout(null)
```

Il componente in questione, non possedendo quindi un *LayoutManager* disporrà tutto lo spazio disponibile assegnandolo al primo Component figlio. Ma fai prima a provare e vedere i risultati che a capire quello che ho scritto!

Java: input da tastiera

Scusate la banalità della domanda. Conosco bene il C e stò cercando di imparare il java! Per stampare a video uso *system.out*, ma per assegnare ad una variabile un valore scelto da tastiera? Cioè per simulare lo *scanf ("%d", &intero)* del c come faccio in java? Grazie in anticipo.

Lupetto

Risponde briz

Non c'è niente di facile quando è la prima volta che si affronta un problema! Dai un occhiata a questo esempio, dovrebbe darti una idea:

```
String dato=null;
int ris=0;
...
try{
BufferedReader str = new BufferedReader(
new InputStreamReader(System.in));
System.out.println("Prova a scrivere un
valore:");
dato=str.readLine();
System.out.println("Ecco cosa hai digitato: "
+ dato);
}
```

Se poi vuoi assegnare il dato letto ad una variabile intera basta utilizzare questo comando:

```
ris=Integer.parseInt(dato);
```

Non escludo ci siano metodi più pratici, ma a me questo funziona di sicuro!

PER CONTATTARCI

e-mail: ioprogrammo@edmaster.it

Posta: Edizioni Master,

Via Cesare Correnti, 1 - 20123 Milano

Biblioteca

ON LINE

CODEBEACH

Un sito che si propone come guida al codice free e open source con tanto di tutorial su ASP, C++, C#, ColdFusion, Delphi/Kylix, Java, JavaScript, Palm, Perl, PHP, Pocket PC, Python, Visual Basic, e XML.

www.codebeach.com/

DOTNET SPIDER

Un guida online alle nuove tecnologie Microsoft orientate allo sviluppo di applicazione. Esempi, progetti, guide passo passo dedicate sia ai neofiti che ai programmatori esperti.

www.dotnetspider.com/

DELPHI GEMS

Librerie, controlli, news, ed un ottimo forum di discussione sul linguaggio RAD per eccellenza: Delphi

www.delphi-gems.com

PROGRAMMARE MS VISUAL BASIC.NET (VERS. 2003)

È una vera e propria guida di riferimento, di ben 1376 pagine! Offre una trattazione ampia e approfondita, completamente aggiornata, di Microsoft Visual Studio .NET 2003. Ricco di consigli preziosi, esempi pratici di codice e best practice, questo libro fornisce le informazioni necessarie per creare velocemente potenti applicazioni Win32 e soluzioni scalabili per il Web, compresa l'interazione con il common language runtime, il multithreading, le applicazioni Windows Forms, GDI+, Microsoft ADO.NET, i Web Form Microsoft ASP.NET e i Web service XML. Un'attenta analisi del testo vi permetterà di aumentare la produttività del linguaggio VB.NET, grazie a nuove caratteristiche e tecniche per scrivere codice più veloce e più affidabile. Uno strumento di orientamento professionale che accresce le competenze necessarie per produrre soluzioni sofisticate per Microsoft Windows e per il Web.

Difficoltà: Medio-Alta • Autori: Francesco Balena • Editore: Mondadori Informatica http://education.mondadori.it • ISBN: 88-04-53078-2 • Anno di pubblicazione: 2004 Lingua: Italiano • Pagine: 1376 • Prezzo: € 75.00

BEGINNING .NET GAME PROGRAMMING IN C#

Grazie a questo testo potrete sfruttare al meglio le potenzialità di .NET per creare giochi divertenti e completi. Lo stile di scrittura è semplice e vi introdurrà alla programmazione in C# per sviluppare giochi in modo veloce. Il testo include anche un'introduzione alle DirectX9 e ad altre caratteristiche avanzate del .NET che vi consentiranno di utilizzare animazioni e suoni. Il libro esamina, a titolo di esempio, cinque titoli in modo da offrire un approccio quanto più variegato alle diverse situazioni di gioco. I codici di esempio di riferiscono a giochi completi, quali ".Nettrix" , ".Netterpillars", "River Pla.NET", "Magic KindergarteN", "D-Infect", "Nettrix II" (per Pocket PC), e una versione del classico gioco "Spacewars". In pochi passi scoprirete come trasformare un gioco 2D in 3D, come creare personaggi con l'intelligenza artificiale, come creare giochi eseguibili in rete e tanto altro ancora…

Difficoltà: Bassa • Autori: David Weller, Alexandre Santos Lobão, Ellen Hatton Editore: Apress /www.apress.com • ISBN: 1-59059-319-7 • Anno di pubblicazione: 2004 Lingua: Inglese • Pagine: 468 • Prezzo: $ 44.99

PROGRAMMARE MS OFFICE EXCEL 2003 CON VB FOR APPLICATIONS E XML

Il testo ideale per gli "smanettoni" dei fogli elettronici che vogliono approfondire le potenti funzionalità di programmazione a applicarle ai dati. Si tratta di una guida avanzata che fornisce tutte le informazioni necessarie per automatizzare i fogli elettronici, scrivere funzioni e procedure e creare soluzioni aziendali personalizzate. Uno sguardo completo sulla programmazione con Microsoft Visual Basic for Applications (VBA), mediante l'utilizzo del modello a oggetti di Excel, che vi permetterà di modificare e personalizzare gli oggetti di Excel, sfruttando le nuove funzionalità XML e integrando le funzioni di Excel con altri programmi di Microsoft Office e del Web. Nel CD allegato al libro c'è il codice di esempio, una versione di prova di "Spreadsheet Assistant", un eBook completo, una varietà di risorse per Excel, dimostrazioni e altro ancora...

Difficoltà: Medio-Alta • Autori: Curtis Frye, Wayne S. Freeze, Felicia K. Buckingham Editore: Editore: Mondadori Informatica http://education.mondadori.it • ISBN: 88-04-53451-6 _ Anno di pubblicazione: 2004 • Lingua: Italiano • Pagine: 640 • Prezzo: € 65.00